WINDCREST®/McGRAW-HILL

2nd EDITION

DOS® BEYOND 640K

- Updated coverage includes:
  - —DOS 5
  - —ISA, MCA, EISA
  - —Windows multitasking
  - —PC-MOS/386, VM/386, & other DOS alternatives
  - —DESQview
  - —DOS extender technology
  - —XMS memory management option

James S. Forney

DOS®

Beyond 640K

2nd Edition

DOS® Beyond 640K

2nd Edition

James S. Forney

Windcrest®/McGraw-Hill

SECOND EDITION
FIRST PRINTING

© 1992 by **James S. Forney**. First Edition © 1989 by **James S. Forney**.
Published by Windcrest Books, an imprint of TAB Books.
TAB Books is a division of McGraw-Hill, Inc.
The name "Windcrest" is a registered trademark of TAB Books.

Printed in the United States of America. All rights reserved. The publisher takes no responsibility for the use of any of the materials or methods described in this book, nor for the products thereof.

Library of Congress Cataloging-in-Publication Data

Forney, James.
DOS beyond 640K / by James S. Forney. — 2nd ed.
p. cm.
Rev. ed. of: MS-DOS beyond 640K.
Includes index.
ISBN 0-8306-9717-9 ISBN 0-8306-3744-3 (pbk.)
1. Operating systems (Computers) 2. MS-DOS (Computer file) 3. PC-DOS (Computer file) 4. Random access memory. I. Forney, James. MS-DOS beyond 640K. II. Title.
QA76.76.063F644 1991
0058.4'3—dc20 91-24629
CIP

TAB Books offers software for sale. For information and a catalog, please contact TAB Software Department, Blue Ridge Summit, PA 17294-0850.

Acquisitions Editor: Stephen Moore
Production: Katherine G. Brown
Book Design: Jaclyn J. Boone
Cover: Sandra Blair Design, Harrisburg, PA WT1

To Sheila

Contents

Preface xiii
Acknowledgments xv
Introduction xvii

Chapter 1. The unexpanded system 1
Physical limits of the system *2*
The physical machine *5*
Life beyond 640K *7*
The operating system *10*
Evolution: a two-way street *12*
What else is in there? *13*
Out of hiding *13*

Chapter 2. At the heart of things 15
The ubiquitous 8088 *16*
The 80286 *20*
The supercharged 80386 *23*
Above and beyond: the *i*486 *29*
An SX version of the *i*486 *30*

Boosting performance even more . . . sometimes *30*
A ticking clock *31*
There's more *33*

Chapter 3. The new breed 35
A brave new world *35*
Games anyone can play *36*
The revolution almost no one noticed *40*
Meanwhile back at the ranch *41*
To the beat of a different drummer *42*
And in the center ring . . . *45*

Chapter 4. Expanded memory 47
In the beginning *48*
A little dinner music, please *48*
Using expanded memory *52*
Expanded memory that is—but isn't—what it seems *53*
Only a stopgap *54*

Chapter 5. EMS memory 55
The 4.0 backfill: mapping conventional memory *58*
But not for everyone *59*
The sheep from the goats *60*
Registers: real and fake—and often missing *60*
Bus speed *65*
The bottom line *66*

Chapter 6. Stealing the store 67
A party of volunteers *70*
DOS 5's HIMEM.SYS and company *71*
Bigger than life *75*
Fragmentation *76*
Declaring open season on the BIOS *78*
For want of a map *79*
It went where? *81*
Stealing still another 64K—maybe even 96K *81*
Don't count the 80286 out yet *83*

Chapter 7. Extended memory and new frontiers 85
A quick review *86*
Protected mode *87*

Bottoms up *89*
Enter the DOS extender *90*
Lotus put it all together—just like 1-2-3 *92*
Canned answers for uncanny problems *95*
What's a VCPI? *98*
DPMI: a light in the window *99*
In these muddy waters *100*

Chapter 8. DOS's mysterious "extra" 64K 103
With a little sleight of hand *104*
Fool's gold *107*
Gift or Trojan horse? *108*
Digging for gold the old-fashioned way *109*

Chapter 9. Chairmen of the board 113
The changing face of management *113*
Duos, quartets, and one-man bands *114*
New directions *115*
Getting down to brass tacks *117*
Super specialists *118*
Gentlemen, start your engines *119*
Me too *121*
Exceptions to the rule *122*
Shots in the dark *123*
Getting downright pushy *123*

Chapter 10. Two times 5.0 127
Left foot, right foot *128*
The Microsoft connection *128*
A cut above *130*
Just having EMS is not enough *130*
From digital research: DR DOS 5.0 *134*
More than just a matter of semantics *135*
Configuring your system on the fly *137*
Choices, choices *138*
Don't send a boy to do a man's work *139*

Chapter 11. Entry-level answers 141
More room with HeadRoom *142*
Context switching: Carousel's revolving door approach *145*
Task swapping under the DOSs *146*

Task switching lurking in the background *148*
FAX boards *148*
Spoolers and buffered output *149*
Peripheral sharing *151*
It all helps—but is it enough? *151*

Chapter 12. DESQview and the age of multitasking 153
The heretics *154*
A textbook 4.0 EMS case study *156*
Stick to the straight and narrow *158*
To build a better mousetrap *160*
Negative overhead *161*
Worlds in collision *163*
Windows' windows *164*
The tortoise and the hare *164*
For power users and beyond *165*
DESQview X *166*

Chapter 13. Windows 169
About a ton of memory *170*
A three-windows world *171*
What Windows 3.0 provides, and what it doesn't *172*
QEMM *173*
386MAX *174*
ALL CHARGE 386 *176*
hDC FirstApps *176*

Chapter 14. Beyond the real 179
If it will run on any 8086 machine . . . *180*
No virtual clock *181*
VM386: multitasking and more *181*
A megabyte for everyone *183*
A clean environment *185*
Split personality *187*
Virtual machines do not have virtual crashes *188*
More than just one pretty face: the multiuser option *189*

Chapter 15. The MDOS multiuser option 191
The virtual machine again *192*
Multiuser VM386 *193*
DR Multiuser DOS *195*

The double whammy *196*
Idle power *199*
PC-MOS *200*
Only the tip of the iceberg *204*
What you don't see *205*

Chapter 16. Keeping up, or trying to 207
Supercharging that old 8088 *208*
The 286 dilemma *210*
The ALL 386SX: a new approach *210*
The empire strikes back: Intel's SnapIn 386 *211*
Charge it with the ALL ChargeCard *212*
Adopting a new mother *214*
The interleave factor *217*
Superfast "disks" and disk caching *219*

Chapter 17. The ultimate upgrade 221
Striking a happy medium *222*
Cash and carry *224*
Down memory lane *225*
Ladies-in-waiting *226*
Don't miss the bus *226*
What's wrong with the old bus? *228*
Beyond "advanced technology" *228*
Putting the pieces together *231*
Recycled oldies *233*
No no no! *233*

Chapter 18. Crash course 237
The high road *239*
Looking in high places—even on an 8088 *240*
On a higher plane *242*
Where did that come from? *243*
Any old port in a storm *246*
Software crashes can be just as hard *246*
Detectives on a disk *249*

Chapter 19. Parting shots 253
Ship of fools *254*
To stretch a point *254*
The more things change *255*
The ultimate shell game *256*

APPENDICES

A. Advanced programming functions available under LIM 4.0 EMS 257

B. Special considerations for mapping LIM 4.0 EMS memory 261

C. Using bootable floppies to test new configurations 265

D. The basics of hexadecimal 269

E. Addresses for the software developers 271

Glossary 273

Index 277

Preface

So much has happened since the first edition of this book was published that many of the questions asked then have long since been answered. Someone recently observed that, in the period since the introduction of the IBM PC, our use of memory has increased tenfold every five years. We went so quickly from just 64K in those first machines to 640K as the standard in later models.

We had reached that tenfold figure less than five years after the introduction of the first PC; however, the 64K figures predates that machine. The five year estimate is pretty close. Another five or so years later, we now have seen another tenfold increase in our needs. To use Windows 3.0 you really need at least 2 Mb; to multitask you'd better add another 2 Mb or 4 Mb more. To do real multitasking effectively in DESQview, VM386, or any one of several multitaskers, you probably need at least that much memory.

Not only has our appetite for memory become almost insatiable, but—with this appetite, a new genre of hardware, and new, more powerful applications software—we must change the way we think about memory and how we access it.

Even DOS itself has changed with the introduction of two powerful 5.0 releases from the two key players in the game (old rivals Microsoft and Digital Research), plus a third if you include the new DR Multiuser DOS 5.0, also from Digital. Each DOS addresses the challenge of our exploding needs in its own unique way. Although there is nothing truly novel in the changes any of these three—or any of the several look-alikes—have made, they do focus attention on the issues in a somewhat different light.

In the reflection of that light and the many changes in the issues since this book first was published, we look now not just beyond 640K but to the issues and the answers as we look ahead to DOS's second decade.

Acknowledgments

Very special thanks once again to Gary Saxer of Quarterdeck Office Systems (Desqview). Without his patience and insights, this project might have fallen short of expectations.

Special thanks also to the following (not in order of their contributions): Paul Tarlow at Qualitas, Inc.; John Barrett at All Computers Inc.; Cristy Gersich and Eric Straub at Microsoft; Dirk Smith at Phar Lap; Roger Kasten at Newer Technology; and Evan Lim at Intelligent Graphics Corporation.

Thanks to the following companies that, in addition to those above, still represent only a part of the many companies and individuals that generously furnished hardware, software and/or expertise specifically for use in preparing this book and in shaping its direction:

ALR
ALL Computers Inc.
AST Research
Borland
Digital Research, Inc.
Fifth Generation Software, Inc.
Hauppauge Computer Works
Hayes Microcomputer Products, Inc.
Helix Software
IBM
Intel
Intelligent Graphics Corporation
Lotus Development Corporation
Merrill and Brian Enterprises
Software Link, Inc.
V Communications, Inc.
Xyquest, Inc.
Zeos International, Ltd.

Introduction

This book is titled *DOS Beyond 640K* because it follows in the footsteps of an earlier book, a quite different book (originally called *MS-DOS Beyond 640K*) that was devoted to that premise. This time around, I have dropped the "MS-" from the title, not to play down or belittle the tremendous efforts Microsoft has devoted to upgrading MS-DOS, but rather in recognition of the fact that users are no longer dealing with a one-DOS world.

I probably could have dropped the "640K" as well and simply called it *DOS Beyond* because the technology has gone so far beyond at this point that a finite number—especially so low a number—is hardly applicable. Yet, the roots of DOS will always lie below 640K, and so that number will remain significant.

In any event, it seems certain at this point that DOS is here to stay—at least through the foreseeable future. If DOS as it exists today does not survive, something close enough to be compatible with the applications software now run under DOS will take its place. However, as the issues and the answers and the focus of the industry itself have changed dramatically, this second edition has changed with them in an effort to keep pace with a technology industry that is at once predictable to some extent but at the same time highly volatile as extended memory, rather than expanded memory, takes center stage.

There is more memory available for everyone—cheap memory—which is fortunate as memory demand increases exponentially. However, the pillar of compatibility, the very cornerstone of DOS, has crumbled. The world of DOS has stratified and will continue to become more so at an accelerating rate. This comes out of the realization that the world could not forever wear the 8088 around its neck like the proverbial albatross.

The coming of the 80386 and the *i*486, which brought 32-bit power, was the engine of the change, yet there was no other operating system on the scene to take the place of DOS. DOS, however, is slippery and resilient. With the advent of the DOS extender, it got yet another lease on life.

It is ironic, as desktop technology leaps forward (no longer measuring its stride in kilobytes or even megabytes but now in gigabytes), that DOS, a veritable dinosaur, should manage to survive at all—not only just survive but thrive. It is an inherently slow 8-bit 1 Mb operating system in an age of seemingly absurd memory limits and clock speeds.

However, life is full of ironies. So, today users have DOS—essentially the same old 8088-compatible DOS they've always had except for some embellishments and for another player (Digital Research) in the game. There is another DOS—the exact same DOS, but running on an 80286 or higher systems playing host to DOS-extended programs, providing them a gateway to protected mode.

Even Windows 3.0 and up can step right over DOS to live and work by preference out in protected memory. While Windows still will run on lesser systems, there is no comparison. This situation has only further crystallized the issues.

Windows was not compatible with DOS extenders as it was first seen, or even with the VCPI (Virtual Control Program Interface) specification. Without some form of mutually accepted interface, real mode programs and DOS extended programs cannot coexist. However, rather than comply with that standard (which was generally accepted by the industry), Microsoft drew up its own, the DPMI (DOS Protected Mode Interface), which, after attracting something akin to a lynch mob, Microsoft then threw open to industry participation in revising.

Out of that evolved a quite different DPMI specification, which addressed a number of shortcomings in both the original DPMI and VCPI specifications, setting the stage for a whole new era. Regardless of the ultimate fate of Windows—which despite the hype, has hardly gotten off to a spectacular start in terms of actual user acceptance—its greatest legacy might be the DPMI specification that it fostered.

With all this, EMS (expanded) memory—once the darling of the industry—has been relegated to a lesser role. However, while no longer in the spotlight to the extent that it was, the technology it spawned—particularly the technology for mapping memory to unused DOS address space above 640K—has played a major role in taking DOS and look-alikes beyond another new frontier to host new multiuser systems for the burgeoning new MDOS (Multiuser DOS) market.

Yet, you should not count EMS memory out by any means. As long as there are 8088s out there—and they're still selling machines—EMS memory will always be the only way those users can break the bonds of DOS's old 640K, as long as there are users running anybody's single-user DOS on the 80386 and *i*486 (and increasingly on 80286s), even with the use of better supporting chip sets that support mapped memory and as long as the needs of millions of users can be met by DOS.

Indeed, both MS-DOS 5.0 and Digital's DR DOS now provide expanded memory (EMS 4.0) emulation (Microsoft for 386 and higher systems only, Digital starting at the 80286 level) with powerful new memory managers that, though lacking in sophistication and convenience, rival most of the better third-party memory managers for raw power.

But as it has been increasingly easy for the rapidly expanding base of 80286 and higher users to cash in on the benefits of new memory technology, there also has been increasing competition for the useful memory gained, particularly in upper, or reserved memory (640K to 1024K), and the High Memory Area (that first 64K of extended memory that DOS can still access in real mode). So, while it has gotten easier, users also are at a point where they need to understand the issues where choices must be made and not simply take the path of least resistance.

It is to this end that this book was written. To this end, I have tried to put things in perspective—into the perspective of the times—as DOS moves on into its second decade.

In this book, I will discuss both the hardware and the software. To demonstrate the various areas as I explore them, I will focus on a number of specific brand names, often on specific products. All are presented solely on the basis of my own hands-on experience. Products that did not perform for me are not included in this book.

This is not to say that you should infer that, if it isn't in this book, it's unworthy. It is true that I specifically omitted mention of some products on that basis. However, while I have tried to bring the most significant to your attention, there are many products—and surely many good ones—I did not examine.

This does not guarantee that every software package and every bit of hardware mentioned will absolutely work for you. At one point, for example, when it seemed that I was having more than my share of software failures on a premium-priced 386 that I was using for a test bed, I finally traced the problems to a faulty BIOS design, which the manufacturer corrected only after over a year of hassle (during which time I had to buy another machine to finish this book).

So the work goes on. I can only hope that my efforts make your work go easier and the time you spend more productive.

1

CHAPTER

The unexpanded system

Before jumping into the middle of things, you need to understand a little bit about the basic underlying system—be it a PC, XT, or AT. (For classification purposes both 80286 and 80386 systems generally are considered to be AT-type systems despite the major differences, which we will be dealing with in detail in this book.) Unlike a number of excellent books, this book does not cover the anatomy of the original PC and its immediate heirs. I'm not even going to give much time or thought to what you ordinarily do in the old familiar 640K except to show you how you can squeeze much more usable space out of that 640K on 386s and AT-type systems. For real performance, I'll show you how to swap most of the 640K—running code and all—for big chunks of expanded memory.

This book primarily looks into the hidden nooks and crannies you might never have known were there or cared about before (except for expanded and/or extended memory). Also, I will explain how to push DOS beyond its original 1 Mb (640K for the user) limitations and make it embrace extra megabytes without undergoing major changes that would make it incompatible with millions of existing PCs and compatibles (possibly your own) and hundreds of millions of dollars worth of installed software.

I'll start with a rather general overview and then move quickly to explore the areas that should be of greatest interest to users of expanded and extended memory. I hopefully will clear up some of the confusion surrounding the different types of memory. However, first things first: the basic system, the foundation.

Physical limits of the system

Physically there really are no practical limits to how much memory you can add to your existing system. As long as you have some way to connect them electrically, you could just keep adding and adding and adding and adding. A gigabyte? Sure. No problem, provided we added a bigger power supply, etc., to support the extra memory chips.

In the real world, however, we have some real-world problems. You could keep on adding chips, but beyond a certain point, your computer could no longer find them. You could see them, point your finger at them and yell, "There, dummy, there!" at your computers. As far as the computer is concerned, however, they simply wouldn't be there. What's more, every time the computer looked, it would find the exact same chips and overlook the exact same group beyond that certain point.

The problem is that a computer can't see. It only can detect or not detect. It's similar to a mouse in a maze. There's only room for so many tunnels and passages to connect to the central chamber where the mouse lives. There might be other tunnels and passages and they might be full of cheese, but if they can't connect, then the mouse can't find them or the cheese.

The mouse—not to be confused with the hairless computer variety—is like the microprocessor chip. In our computers, the passages and tunnels are like the pins that connect electrically to whatever else is out there. The old 8086 on which this dynasty was founded had only 20 address pins—whatever other pins were sticking out of it served other functions. Each address pin, when combined with other address pins, could look to many addresses.

The magic number two is the binary base of all computer operations. In this context each address pin can have two states—either it's on or it's not. If we only had one address pin, it could point to either of two addresses—one if it was on, the other if it was not. Two pins, each with two states, can manipulate four addresses, and so on. In the mathematics of computers, a 20-pin chip could address 2^{20} addresses. That product yields a total of 1,048,576 unique memory locations from 0000h to FFFFh, or one megabyte. See Table 1-1.

One megabyte. Beyond that magic number, you can plug in chips until the world is flat. One megabyte of addresses is all that poor old CPU can find no matter how hungry it is for more. (The extra DOS-addressable 64K HMA above 1 Mb is available only to 286 and higher chips.)

Someone is bound to ask how computers manage to manipulate data from 20 address pins using only 16-bit address registers. Someone else surely laid awake nights to figure that one out. The answer is using the method of dividing absolute address locations into two parts: a segment and an offset. The format for such

Address Pins	Possible Addresses
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1,024
11	2,048
12	4,096
13	8,192
14	16,384
15	32,768
16	65,536
17	131,072
18	262,144
19	524,288
20	1,048,576
21	2,097,152
22	4,194,304
23	8,388,608
24	16,777,216
25	33,554,432
26	67,108,864
27	134,217,728
28	268,435,456
29	536,870,912
30	1,073,741,824
31	2,147,483,648
32	4,294,967,296

Table 1-1 The powers of 2 show the relationship between address pins on a processor chip and the mathematically possible range of addresses. These numbers also coincide with other numbers commonly encountered in computer systems, such as 512 bytes to a sector (disk), 1024K in a megabyte, and 64K blocks that actually contain 65,536 bytes. Not just addresses, but virtually everything done with a computer has its roots in this table.

addresses is *segment*: *offset*, or as displayed (in hexadecimal) by DEBUG, something like:

```
0000:0000
```

While we will touch lightly in later chapters on the mathematics of translating absolute addresses to this format to accommodate 16-bit registers and the anomalies that result, these are subjects beyond the scope of this book or the needs of most readers. They, however, are covered amply in other current literature for those who are interested. Each of these hex digit *segment:offset* addresses represents an absolute address requiring only four hex digits. Five actually are required to get all 20 bits but, for most purposes, the last one can be ignored. We are concerned with those absolute addresses, not with the mumbo jumbo of a 16-bit world.

Lest you get the impression that having only 16-bit address registers to work with imposes severe limitation on how much memory can be addressed, consider this fact: Two 16-bit registers working together can, at least theoretically, handle 2^{32} unique real addresses. Even with only 16-bit registers, you're in little danger of running out of memory capacity.

The 80286 and then the 80386 added more address pins. Referring back again to Table 1-1, showing the relationship between address pins and the exponential rate at which adding more dedicated address pins increases the potential number of addresses, you can see how relatively easy it is to add more pins.

The 80286 could handle 16Mb (though other hardware limitations in the original IBM AT restricted it to 4Mb). The 80386 could open up 4 gigabytes, 4000 Mb of address space, thanks to its 32-bit architecture. When every extra address pin you add redoubles what you had before in address spaces, memory capacity adds up awfully fast.

DOS, however, came into being to support the 8086/8088 genre, so therein lies the dilemma. An 80286 runs quite nicely in only a 1Mb limited address space, as does an 80386, though it's a little like putting a 1000 hp engine in a Toyota just to drive down for a loaf of bread. But it works. Both of them work much faster than an 8086/8088 because the 80286 and 80386 can be run with faster clocks. (An *i*486 is even faster yet for reasons besides just clock speed.)

If not for the faster clock speed, you'd hardly notice any difference right away. Most of your old 8088 software runs just fine. Some didn't, but then software upgrades quickly took care of that problem. DOS 3.x came out along with the 80286 AT. The 4.x series, right from 4.0, included some special 80386 support. DOS 3.x was backward-compatible with the older 8088 machines. Theoretically, 4.x is fully backward-compatible back to the old 8088 machine, as well. Initial releases through IBM's 4.01 exhibited some severe backward compatibility problems, including an inability to find or read a hard disk even on older genuine IBM machines. By press time, most of these problems either seemingly had been fixed or fixes were assured. Going in the other direction, the older DOS versions, beginning with 2.x, would run on the new machines.

The degree of compatibility that has been achieved is little short of miraculous, especially in view of various monkey wrenches and assorted obstacles some of the heavy hitters in the game have thrown in along the way. Those of you who might recall the earlier debacles of Apple, which somehow endured a period when nothing Apple made was compatible with anything else, will surely join in a hearty cheer to that.

Unfortunately, the degree of compatibility is the good news. The bad news is that, to maintain that compatibility, DOS and everything that runs under it is still restricted to just 1 Mb of address space. Forget about extended or expanded memory. Anything more than 1 Mb, under DOS, is sheer sleight of hand. Now you see it, now you don't, because it isn't there. It never was, never will be, and never can be, not under DOS.

Is there another operating system available, something beyond DOS? OS/2 perhaps, someday. (That is exactly what I said about OS/2 in the first edition, which says something for how far OS/2 has progressed.) Possibly something else though, something we haven't even seen a glimmer of as yet. All things are possible—or almost all. However, when another operating system comes along and really pushes DOS aside, the now operating system is going to have to have enough things going for it to make abandoning most of our present software and much of our current hardware worthwhile, because that marvelous backward compatibility is going to go the way of the Edsel.

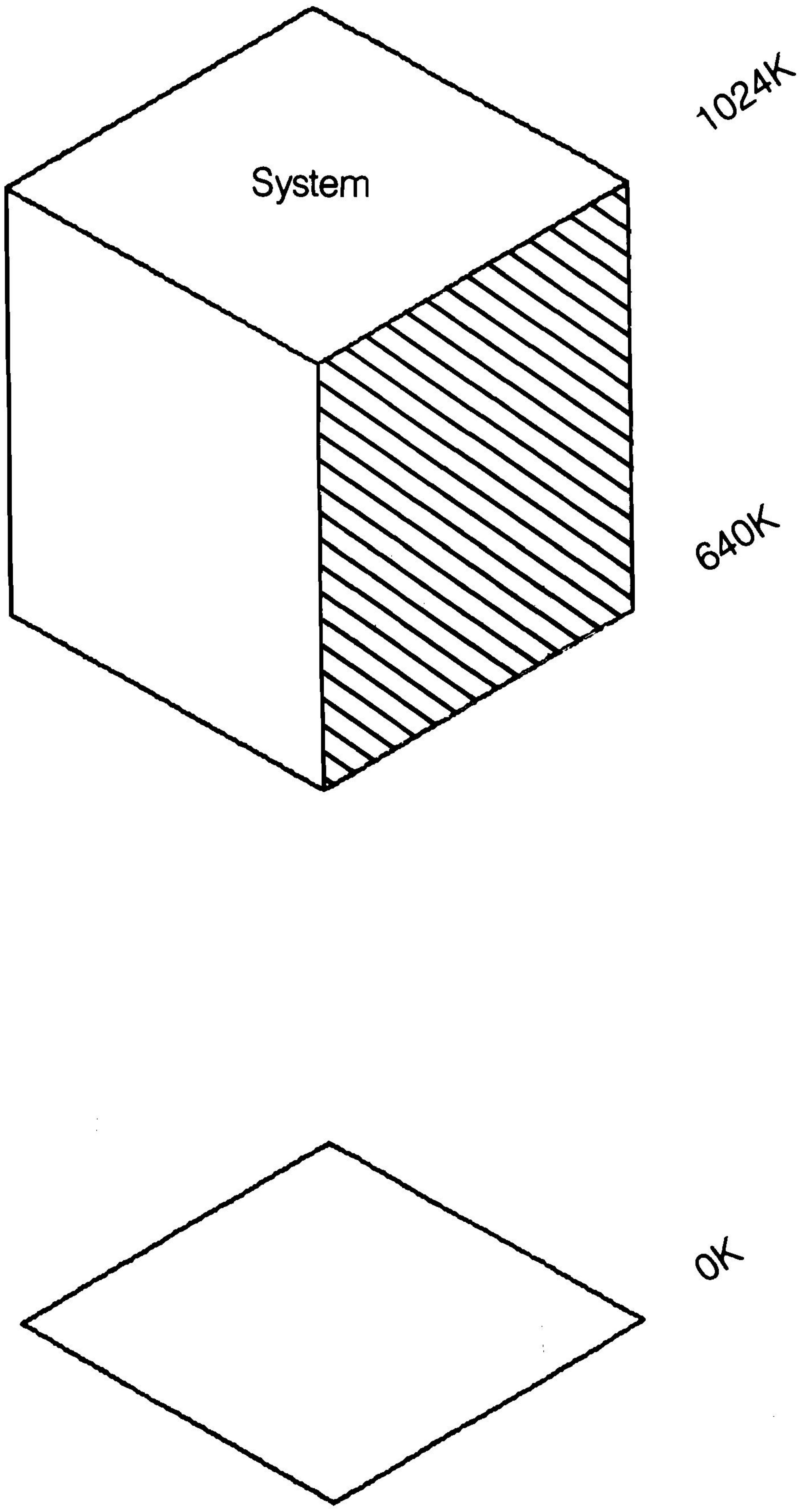
For now there's DOS. For a long time yet to come, it's going to be there. As still new tricks and techniques are being added to DOS's repertoire, you can be sure DOS is going to be around long after something else grabs center stage.

One thing to keep in mind is that there are some things in DOS that cannot change to keep pace with technology. That old 1 Mb bugaboo is one of them. Therefore, as long as you use DOS, if you want to address memory beyond 1 Mb, you've got to somehow fool DOS (and whatever software you're running under DOS) into thinking you're still working within that 1 Mb the old 8086 stuck us with. This trickery is what expanded memory is all about. But before we get carried away with that, however, let's take a look at the physical machine we have to work with to better understand some other issues.

The physical machine

Starting with a microprocessor chip with 20 address pins capable of handling 1 Mb of address space, the PC was conceived—before there was a DOS. This idea came at a time when the idea of 1 Mb of memory for a desktop machine seemed so incredible as to be ludicrous. Some of the IBM's first PCs were shipped with as little as 16K of user RAM. And such was the state of the art that people actually bought them that way.

In this climate, some critical and irrevocable decisions were made as to what to do with all that address space. It was all pretty academic—similar to the play money in a Monopoly game—but somebody had to do it. Therefore, 640K was generously assigned, preposterous as it might have seemed, to user applications. A whopping 384K was reserved for system overhead, both real and projected. See Fig. 1-1. At that time, they didn't need 384K, or anything like it. These same geniuses, however, didn't even foresee PC users ever even wanting, let alone demanding such things as floppy disk drives.



1-1 Suddenly free of earlier 64K total-system limitations, designers of the PC grabbed 384K of the 1Mb address space for system use.

We're really lucky we got 640K out of the deal when you consider the thinking that went into it. 640K, however, was what we got, and the engineers immediately began slicing up the rest of the pie for use by the system, which they understood about as well as they understood the market that would quickly develop and give the PC a life of its own. The market would have been completely out of their control except that they had made those critical decisions and the die was cast.

If not for a bunch of greedy engineers grabbing all that memory that they didn't need, there would be no such thing as expanded memory and the PC might well have gone the way of the high-button shoes. That critical 384K block of memory reserved for the system—memory beginning at A000h (upward from 640K)—is what we will be dealing with primarily in this book.

Through a piece of that hoarded space above 640K that IBM and the other powers-that-be finally released, we now can access not just little leftover scraps—very useful scraps we'll find when we focus on 80386s—but megabytes of expanded memory via sophisticated techniques. Some maybe aren't so sophisticated, but they work anyway.

Unfortunately, in the intervening years, many people eyed some of the unused address space up there above A000h and decided that IBM was never going to use it. Having reached that decision, they made the further assumption that, if IBM was never going to use it, no one would ever notice if they just sort of moved in on it. So they did. Some of the most respected names in the industry are among the squatters that now are so firmly entrenched that there is no way to move them to where they belong. In later chapters, I will deal with the reality of their existence and how to cope in a less than perfect world. For now, let's take a closer look at just what there is, or is supposed to be, up in that mystical top third and a little more of our system.

Life beyond 640K

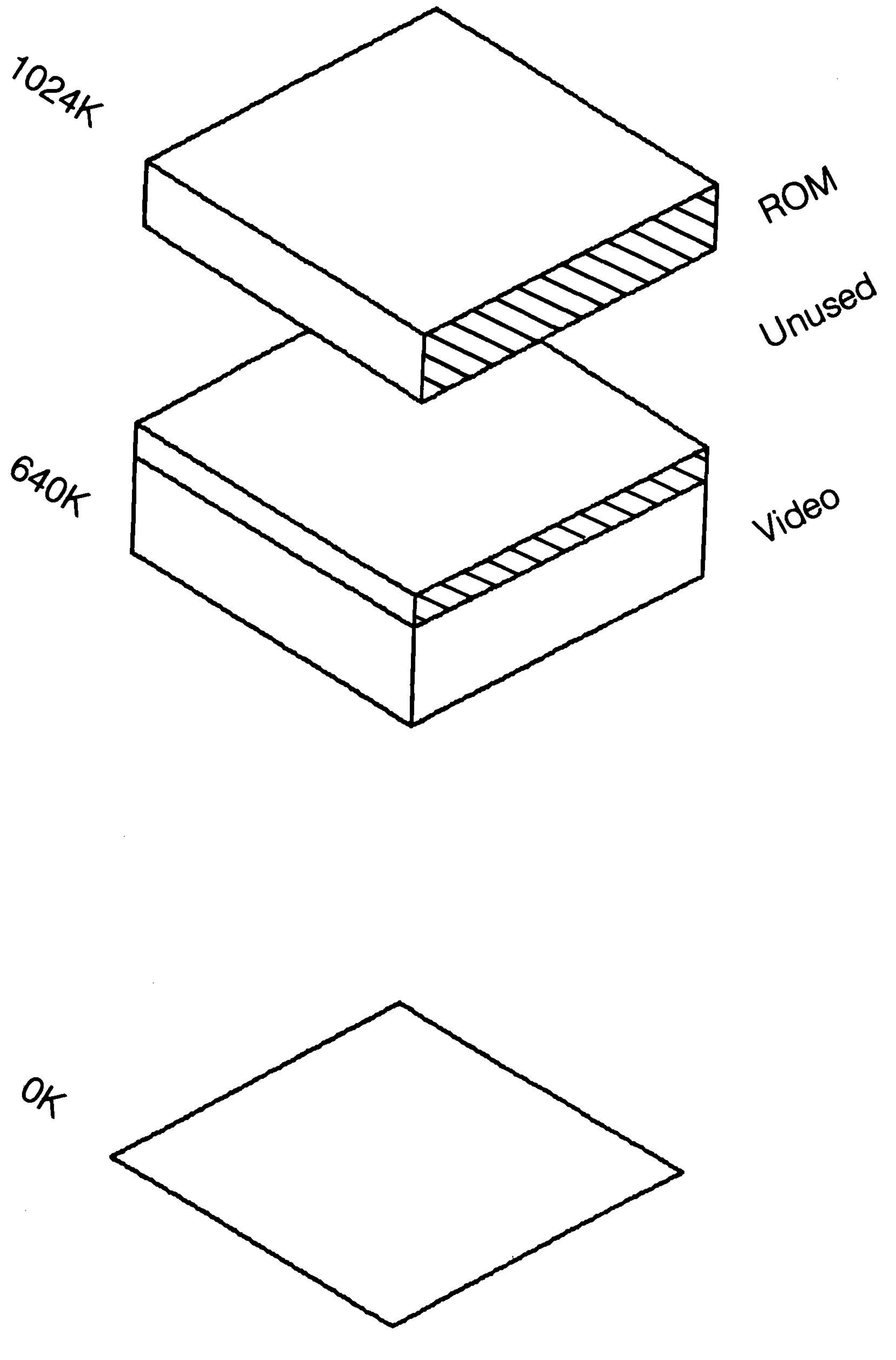
Most users probably know more about the surface of Mars than about what goes on above 640K. To a point, that's fine. If we all got bogged down in the minutiae of our technology, there would be little time to put it to work for us. Still, as we push our computer systems far beyond the imagination of their original designers and come closer to achieving what until now have been only theoretical levels of performance, we walk an even-narrower line between perfection and utter chaos. The more we know about what goes on in there, the closer we can go to the brink without falling off.

As you have seen, 640K is just an arbitrary number. From day one, the original PC was a megabyte machine (1024K). The other 384K has always been in there, even before DOS was developed. The ROM that the first PCs booted from had its base address up at F000h (960K) and ran on up to roughly FFFFh (1 Mb) as does the ROM installed in almost every new machine since then. When DOS came along, geared to a 1 Mb machine, it was, and is, a 1 Mb system. We, however, are getting ahead of ourselves, because the origins of 640K go back before there was a DOS.

Let's start by taking a look at just how the engineers decided to divide their share of the pie. In Fig. 1-2, I've narrowed the focus to just the address range above 640K. You can see the areas pretty much as originally assigned.

You can see there is a lot of space above 640K that isn't being used, but programs can only load and run in contiguous, or unbroken, blocks of memory. Sitting right up there blocking access to anything above 640K beginning at 4000h is the 128K set aside for video use—and the beginning of the legendary 640K limit.

Do you have any idea how little actual memory is required to display monochrome text? Bit-mapped graphics is another matter, but in the monitors of the day, about 4K was typical (based on 25 lines of 80 characters using conventional character generation techniques). The exact amount of memory mapped and available for video usage varies somewhat between various video cards and different manufacturers. The original IBM monochrome adapter card mapped memory



1-2 Of the original 384K set aside for system use, only the top 64K and the bottom 128K actually were assigned for specific usage (the top for ROM and the bottom for video). Of that latter 128K, many monitors needed no more than the 32K at the top of the block. That left as much as 96K contiguous to the 640K set aside for users completely wasted, along with 192K in the middle. Over the years, system usage of these areas has changed, but there still is much unused space above 640K.

between B000h and B100h, while its CGA adapter mapped a block four times that big beginning at B800h.

Manufacturers made no pretext about needing that much space. By starting actual assigned monochrome monitor addressing at B000h (704K), they left an extra 64K of contiguous memory space temptingly attached to the 640K of user memory addresses. In practice, most of the real monitors in existence started 32K higher yet (somewhere up around B800h), leaving 96K of contiguous address space untouched.

This contiguous space was not wasted on early hackers, especially as user memory demands crept up and started nibbling at the invisible 640K barrier. Note, however, that I keep talking about address space, not about actual installed memory. There was generally nothing at those addresses, like a new subdivision that still had some empty lots and no real owners in sight.

These empty addresses were similar to motherboards that came from the factory with only one bank of memory chips installed but several rows of empty sockets that, with IBM's PC, would finally support up to 256K on the motherboard. However, there weren't necessarily even sockets up there to plug into, just addresses for things that largely didn't exist. The addresses, however, were assigned, available, and legitimate and the computer would recognize any device, within reason, that reported itself installed at those addresses at boot time.

Even with the 8088 machines of the time—machines that, unlike today's 80386s and up, could not remap unused address spaces above 640K because of limitations of the processor itself, plugging in more user memory was a fairly easy and straightforward trick, provided you could set the memory to start at the addresses higher than A000h. Some vendors of what then were often referred to as "expansion boards"—nothing to do with expanded memory under the LIM specifications—even began providing at least optional support for a minimum of 64K and, in some cases, up to 128K of scavenged memory.

Because there was no contiguous 128K block above 640K, methods of making still more memory available for user applications were generally crude and often required disabling parity checking to work at all. As a result, 704K became more or less a defacto practical limit. Some clone manufacturers actually made 740K standard on their machines before the EGA display came into common usage.

Even the introduction and popularity of the IBM Color Graphics Adapter (CGA) caused no problems. The CGA, a crude device with relatively modest memory requirements, installed itself at the top end of the 128K reserved for video.

The introduction of the EGA display, however, changed the situation abruptly. With much greater address/memory requirements than their predecessors, EGA displays plopped themselves into that previously "reserved for video" space starting at A000h. This space, after all, supposedly was reserved for video usage.

Unfortunately, no two things can occupy the same space at the same time, even if the space is actually only an address that might belong to something intangible. In computers, the address is important. It's fine as long as nothing else actually is using it; however, when any two things, hardware or software, try to use the same address at the same time, something is going to crash. Which peripheral has the right to access the address is irrelevant. The result is still the same; the system will crash.

While it is not difficult to deal with intellectually, this principle is something we must come to terms with to understand what is going on above 640K, at least as it impacts on the work we do. Therefore, you should put that concept at the top of your list. You will be meeting that one again. In a later chapter, you will learn how to recover your system, when—not if, but when—things do collide up there.

I have described briefly what has happened historically in the bottom one third (128K) of the memory reserved for system use. There is a tale if intrigue—claim jumping, and even what could sometimes best be characterized as proprietary sabotage—attached to usage of most of the remaining 256K, as well. The pattern is pretty much the same, and you can't always tell the good guys from the bad by the color of their hats.

The operating system

For the sake of understanding DOS as it relates to Extended and Expanded Memory, the operating system must be broken down to the basic modules or building blocks that go together to make it up.

Users talk about DOS as *the* operating system, as if it were the total package, as if without DOS we would have nothing but a bunch of ill-assorted chips and hardware totally incapable of doing any kind of useful tasks. In practice, DOS has increasingly assumed that reputation by association. Everybody knows that it is impossible to run PC-type computers or compatibles without DOS or something very much like it.

DOS actually provides only a part of the actual operating system. For the proof, we have only to look back to IBM's original PC. And not just the first few that came off the line with only 16K of memory, either. Not only the oldies, but also the genuine PC (right up to the time IBM quit making them) did not need DOS to run. You can turn one on—no hard disk or floppy drive was installed—and in a few seconds the system will boot, the monitor will be alive, and the keyboard active and ready to go to work. Without DOS or any external "system," the system can even save programs and data and/or retrieve them from an external mass storage device.

Few users ever really used their old PCs without DOS and a disk drive—or surely not for long. The point is that the PC had built into it a complete, free-standing operating system capable of setting down the operating rules and managing system I/O: keyboard, monitor, and a mass storage device. It even included BASIC that loaded automatically from the ROM when the system was booted. (Putting the BASIC kernel in ROM still is a distinguishing feature of IBM machines.) That free-standing system was called the BIOS (Basic Input/Output Services).

If your PC has a floppy disk drive and you have a copy of DOS, you could boot using DOS. You would not be booting with DOS instead of the built-in operating system, rather in conjunction with it. In fact, the boot sequence first looked to the internal operating system for everything it really had to have to run, then to DOS for whatever little extras DOS might have to offer.

One of these extras is the floppy disk drive. The original PCs didn't really need one. In fact, the creators of the PC really didn't expect most users to even want one of the then-expensive floppy disk drives. The engineers envisioned the cassette tape recorder as the most popular mass storage device for the PCs, so that was the device they provide I/O services for. The engineers also included an appropriate set of MOTOR commands for the special version of ROM BASIC that was built into the PC (not surprisingly nicknamed CASSETTE BASIC).

If you wanted to use a disk drive instead of a tape drive you needed a disk operating system. You needed DOS (Disk Operating System). You needed DOS, which you had to get from a disk, in order to use a disk. Some data and instructions were needed even to read other instructions from the disk. These instructions had to be put somewhere where even a dumb system would stumble over them. They also had to be abbreviated enough so one quick gulp would provide at least enough I/O instructions to allow the system to suck up the rest of the disk operating system and whatever assorted services related to disk operations it might contain. That is the heart and soul of DOS.

That first gulp, generally referred to as the boot record, is always located on the first sector of the first track of a floppy disk and always on side 1 if the disk is double-sided. (On a hard disk the boot record is located in the first sector of the DOS partition.) If the information is any place else on the disk, the half-awake computer can't find it. Even if the computer accidentally stumbled over it, the system wouldn't recognize the information. At that stage in the boot cycle, the computer still is pretty stupid.

Once the computer has gulped down that first sector of track 0—like that first cup of coffee to get you started in the morning—the system, although still only half-awake, is ready for bigger and better things.

The BIOS is read next. The computer already has a BIOS. One was built-in at the factory. If your computer didn't have BIOS, the computer couldn't even read the boot sector of a disk when it did stumble over the sector. The computer has one BIOS built-in (actually ROM chips that are plugged in), but the system still needs something else. The BIOS information stored on your DOS diskette adds more functions than the built-in BIOS provides. In some cases, the disk BIOS overwrites at least some of the instructions the computer picked up from its built-in BIOS.

The important thing to keep in mind—which will be increasingly important later in this book—is that DOS, or what is normally thought of as DOS, cannot do the job alone. It must rely on and work in conjunction with the most basic parts of the operating system that came built into your computer. Any other operating system that comes along—DOS, XENIX, PC-MOS/386, OS/2, etc.—must ultimately attach itself to this foundation, embrace it, and be completely compatible with it.

That underlying part of the operating system, which must be read into memory during the boot process, usually is contained on a special kind of chip called a ROM (Read-Only Memory) chip. Hence, the computer's built-in BIOS is called ROM BIOS. No functional computer by today's standards can get up and running without some form of built-in instructions, at least enough to get things started when you throw the switch. Without this standard and basic set of underlying instructions, programs could not operate independent of and oblivious to the physical setup of the machine.

The initial DOS, little more than an afterthought, was pretty skimpy. It did add support for disk I/O: single-sided, 8-sector disk format. DOS, however, provided a better disk directory structure than CP/M, the operating system most used in other contemporary computers designed around the then-popular 8080 chip.

The scheme of DOS version 1.0 included such things as file attribute management and showed not only exact file size but also the last modification date. The initial version of DOS also provided for an AUTOEXEC batch file for startup initialization; however, disk services not contained in the ROM BIOS were its primary function.

Most essential services now associated with DOS and taken for granted were yet to come. Today, you so often see the caution: "Requires DOS 2.0 or higher." The seemingly basic services most of today's software requires to run just were not there, not in the ROM BIOS and not in DOS 1.0.

IBM was the only vendor to supply DOS 1.0, but then there were no other players in the game, either. The bandwagon was still under construction; however, the infant, which would soon become a giant, had taken its first faltering steps. Even by the time version 1.1 was released, there already were a few more vendors and, for the first time, there was a parallel MS-DOS available.

DOS 2.0 contained major changes and began looking like a serious operating system. In addition to its early duties, 2.0 offered I/O redirection, pipes, filters (borrowed from UNIX), plus print spooling, volume labels, expanded file attributes, and greater configuration options via a CONFIG.SYS file. Version 2.0 also added an ANSI display driver, allowed dynamic control of memory by programs, program-environment block maintenance, and even user-customized command processors.

DOS was growing up, although it still had, and still has, some serious shortcomings. DOS has managed to mask many of those incompatibilities not only from the end user but also from the software. It has tried to be too many things to different people who are using different and not totally compatible computers.

DOS is slow and always will be. It is tied to an outmoded segmented memory module and, as such, cannot be written to support protected mode operations that are the real key to the future. It cannot deal with memory that is not contiguous. The list of negatives goes on and on. Yet, for many users—most users probably—DOS still is the best solution and will be for some time yet to come.

Evolution: a two-way street

Through the years, even the disk services provided by DOS have undergone considerable change. New disk services have been added as needed, sometimes to support devices neither anticipated or supported by the ROM BIOS. The pocket-sized 720K 3.5" disks were neither anticipated or supported either by the ROM BIOS of older machines or even by the supplementary I/O of any DOS through 3.1. Actually, the 3.5" disks could be used to their full capacity with any DOS from 2.0 or up but only in conjunction with a third-party device driver. Version

3.2 included direct support for 3.5″ disks, allowing older machines to be ungraded easily to include not only these higher capacity disks but also other features and functions not included in earlier versions to help stave off obsolescence.

By the time DOS 3.3 was released, most vendors had incorporated support for 3.5″ disks and some other needed services into the various proprietary routines in their applications. The burden of providing a stopgap solution for the small, high-density floppies no longer fell on DOS. Not surprisingly, DOS dropped those services. The interrelationship between DOS and the many "compatible" computers DOS serves is evolving continually, as both the markets and devices change.

What else is in there?

Most of the DOS services discussed so far are contained in two files that never show up in a directory listing. The DIR command simply can't find them. They are locatable, but they are hidden. One of the file attributes is set so DIR can't find the file. Even if you're a snoop and do find these hidden files, DOS still has another locking mechanism to prevent accidental damage. By means of another attribute, these files are set to read-only.

Generally referred to as the "BIOS module," one of the hidden files is IBMBIO.COM. As the name implies, its primary function is in providing auxiliary I/O services beyond those included in the ROM BIOS routines. The MS-DOS file, IO.SYS, is the equivalent of IBMBIO.COM.

The other, referred to as the "DOS kernel," provides the necessary software interface with whatever applications software you are using. IBMDOS.COM or its Microsoft equivalent, MSDOS.SYS, provides a group of hardware-independent services called "system functions." These include:

- File and record management
- Memory management (conventional memory only)
- Character device input/output
- "Spawning" of other programs
- Access to the real time clock

Applications programs access these functions by loading the appropriate registers with parameters specific to the functions required. These parameters then are transferred to the operating system for execution.

Out of hiding

The part of DOS that users have the most contact with is the command processor: COMMAND.COM. This file is sometimes referred to as the "shell"; however, it should not be confused with the SHELL, or graphic interface, DOS added beginning with version 4.0. To avoid confusion, this file will be referred to only as the command processor, or COMMAND.COM, for the purposes of this book.

The name tells all: COMMAND.COM. It is a complex module that contains not only a number of built-in, or internal, command functions but also other external command functions, including the processing of batch (.BAT) files.

Internal refers to commands that can be executed immediately upon completion of the boot process and the loading of the command processor. These commands are available in the raw system with nothing extraneous loaded; no AUTOEXEC batch file, nothing—just the familiar DOS A:> (or C:>) prompt on the screen. With nothing else loaded on the system, you can copy and delete files, get directory listings, and create, change, and remove directories. These, and other, commands are internal to COMMAND.COM.

Commands such as FORMAT, MODE, and CHKDSK are *external* commands. The command invokes some separate utility program, which must be available either on the default disk and directory or in the PATH or are pathed-to as part of the command. For example:

```
C:> \DOS\FORMAT
```

Look in the directory on the DOS distribution disks. Anything in your DOS directory that looks like the name of a valid DOS command but is an executable file with a .COM or .EXE extension tacked on is the file that is executed (with some help from the COMMAND.COM) when you type the command. All of these .COM and .EXE files are nothing but utilities and are not integral parts of DOS. Most can be erased from your disk without causing any noticeable problem. Add the control and execution of all batch (.BAT) files to this list of COMMAND .COM functions and you've got the whole story—or at least all you really need to know for now.

The important thing to keep in mind, though, is that everything else usually referred to as DOS really is just a useful collection of utilities. Even the new expanded memory device drivers, like 386EMM.SYS (or IBM's XMA2EMS-.SYS and the tag-along XMAEM.SYS for 80386 systems) supplied beginning with DOS 4.0. They are all just utilities, and not even necessarily the best utilities available to do a lot of jobs you might have thought only DOS could do.

Even COMMAND.COM itself is not indispensable. Hewlett-Packard MS-DOS computers have from the beginning been sold with a proprietary screen-oriented shell, called the Personal Applications Manager. Functionally similar to the optional graphic user interface or Presentation Manager supplied with DOS beginning with version 4.0, the proprietary HP version was designed to be particularly suited for use with HP's TouchScreen HP-150 but also to be used with other HP/DOS machines, as well. There also are various third-party command interpreters available—4DOS being one of the best—that can be used to replace COMMAND.COM with any DOS version from 2.0 through 5.x.

2
CHAPTER

At the heart of things

Strictly for the sake of argument we are going to break a computer down to only two component groupings: the *microprocessor*, where all the actual work gets done, and *memory*—conventional, extended, and expanded. Admittedly, this division is a simplistic way of viewing a complex subject; however, for all that is involved in a computer, these two elements are the focus here. The microprocessor is included only because it governs the set of rules memory can run under and how memory can or must be addressed. Because the chip you use establishes the rules, this chapter will start with the microprocessor.

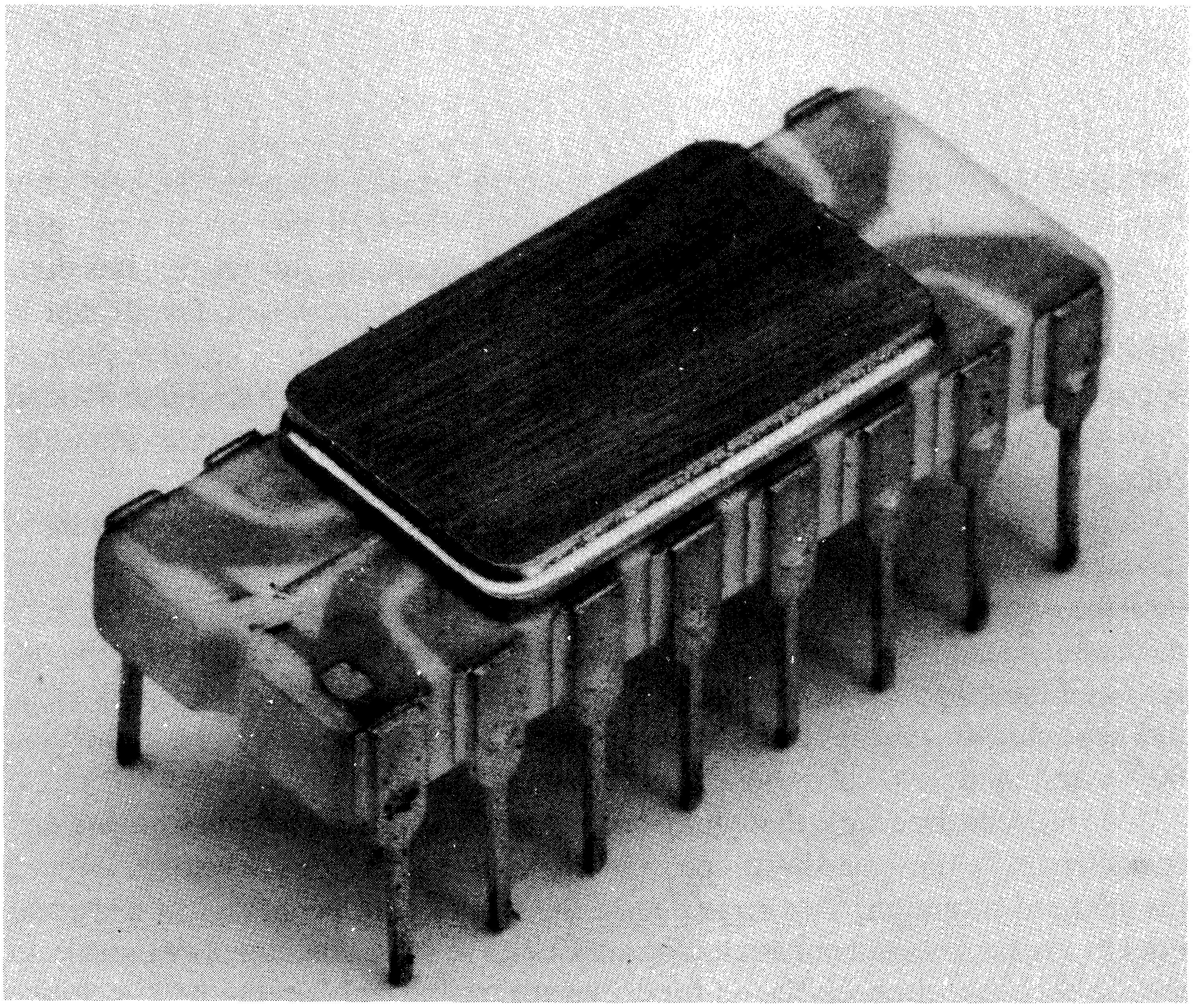
Essentially, this book will be dealing with computers designed around three uniquely different, yet closely related, microprocessor chips: the 8088 that powered the original PC, the 80286, and the 80386 and *i*486. The 80386 and *i*486 are the heart and soul of successive generations in the sense that one evolved from and followed the other. However, just as the airplane did not replace the automobile, the new chips in the industry have not replaced their predecessors, rather the 80386 and *i*486 have only opened new frontiers.

Along with the chips, there have been parallel evolutions of both software and specialized auxiliary hardware—particularly memory devices—developed as need has fostered invention. To a large degree the spectacular technological leaps and bounds are backward compatible and applicable to most computers running with any of the three chips discussed here. We are increasingly seeing both software and specialized hardware, requiring at least a 286 and in other cases, 386 and higher platforms.

Backward compatibility, once considered almost mandatory by many, no longer limits our horizons. Unlike an army, technology cannot slow its pace to lock step with its slowest members. In many ways, however, we still are tied to concepts that are rooted deeply in that way of thinking. Before you plunge too deeply into the total picture, look briefly at some of the differences as they relate to how far each chip can hang on to the coattails of the next before dropping off.

The ubiquitous 8088

Historically, the roots of the 8088 can be traced back to the introduction of the 8008 by Intel in 1972. The 8008 was the first commercially available 8-bit microprocessor. Not much needs to be said here about the 8088 or the dynasty-founding 8086. However, to understand the concept of bits, you really need to go back one generation further to the earlier Intel 4004, which, as the name implies, was a 4-bit chip (Fig. 2-1).



Intel

2-1 Intel's 4004, the world's first microprocessor, is the chip that started it all.

The significance of four bits is that the number four in a binary system (base-two) is the minimum number capable of representing the numbers zero through nine. One bit can count to two, two bits can count to four (two times two), and three bits only to eight. You have to have that fourth bit, making the system capable of counting all the way to 16 ($2^4=16$), to reach a count of 10.

If all you want to do is work with simple numbers, then four bits is fine. The 4004 was used in simple calculating machines. The chip could add, subtract, multiply, and divide and was a veritable wonder of technology compared to the myriad of springs, gears, cams, and cogs of contemporary mechanical office machines.

Then someone got the bright idea of representing the letters of the alphabet, too; hence, the quest for an 8-bit chip. In a binary system, most everything is based in multiples of two.

Eight bits means a possible 256 combinations. That amount is more than enough for 10 numerals, a full alphabet (in both upper and lower cases), plus some of the more common foreign language variants and punctuation. Some special symbols also are included: operands and Greek letters. (Sixteen- and 32-bit machines deal with the same characters found in 8-bit processors; however, the larger processors work much faster.) So Intel then developed the 8008 (really little more than a 4004 with more bits) and, shortly afterward, the 8080 chip, which represented a quantum leap in technology.

These two developments opened up a whole new can of worms. As long as you only had four bits to play with and stuck to simple addition, subtraction, multiplication, and division, no real programming was required. Everything was done in real time; there was no disk storage. Life was simple in a 4-bit world. Not so in the vast new, untamed 8-bit frontiers. As with the hulking mainframes of the day, some sort of programming was required. There, however, was a problem: there was no suitable programming language in existence and there was no easy way to store or load programs even if a programming language had existed.

One group of engineers split off and, deciding that the 8080 was not the way to go, fathered another chip: the Z80. An 8-bit chip with 16 available addresses and a whopping 64K of memory, the Z80 is of special significance—even to us here—because its proponents developed the first rudimentary operating system for a microprocessor chip. Patterned roughly after mainframe systems, the new operating system evolved into what later came to be known as CP/M (Control Program for Microcomputers).

In the meantime, Intel kept refining and polishing its 8080—still largely an orphan—and spinning off new variants, in search of a market share. Intel developed a somewhat similar 16-bit version of the 8080, called the 8086, which evolved further still to become the 8088. Although the 8086 had a 16-bit potential, it found its niche in 8-bit systems that were cheaper to produce. These systems were close enough in design to the earlier 8008 (and, more significantly, to its competing Z80 stepchild) to enable programs written for CP/M to be converted easily to run on it. Although the 8086 allowed faster clock speeds than the 8088, the 8086 was not an immediate success. It found use in the Compaq Deskpro and the AT&T 6300; however, it was tied to an 8-bit bus in both machines and never achieved its full potential. More recently the 8086 has appeared in the PS/2 model 30 and is used in a number of clones.

Although the exact chronology of what was going on behind closed doors is hazy, IBM was working on what would become the first PC (IBM Personal Computer) somewhere in this time span. At this point, these seemingly totally divergent forces start to come together, meet, then split apart again still farther, but not before each played a major role in shaping the role of computers from that to the present. At one time, IBM actually marketed CP/M-86 (a modification of the Z80-oriented CP/M adapted to the 8086/8088 environment) beside DOS as an optional alternative operating system for those first PCs.

DOS actually evolved from another spinoff of CP/M. The original work was attributed to Tim Paterson with a fledgling company called Microsoft, which acquired the rights from another company called Seattle Computer. The PC was still little more than a toy with no clear-cut direction or purpose at this stage.

To maintain even a reasonable degree of compatibility with programs written for CP/M, Intel broke the 1 Mb of address space allowed by its new 8086 and 8088 chips into 64K segments. The segments were 64K because that was the total amount of memory allowed by the Z80 for which the original CP/M had been written. The original PCs were only equipped with 16K (standard), expandable to 64K only on the motherboard before you had to add an expansion board, which fortunately the PC's open architecture allowed.

DOS initially sold beside CP/M-86 for about half of CP/M-86's price and had the additional benefits of an easier, more logical user interface and syntax. DOS's lower price and extra benefits soon effectively pushed CP/M-86 out of the IBM/clone PC marketplace. Unfortunately, during this critical formative period, Microsoft apparently ignored warnings from Intel, which foresaw compatibility problems between the segmented memory model DOS was embracing and the protected mode that coming generations of the 8086 family would run in. In a nutshell, this state of incompatibility is where we are today.

The PC—even the "original"—evolved into something far beyond the wildest dreams (or nightmares) of its creators. Intel kept designing newer and better chips around which two new generations of machines have been built, with still another generation, 80486s, emerging. The more things change, however, the more things stay the same. A 1 Mb operating system with 64K segments rooted in an operating system no longer even mentioned in the same breath as the PC and its faster offspring still is available.

Despite its limitations when compared to the 80286 and especially 80386, the 8088 remains a dependable workhorse capable of coming as close to multitasking as many users need, via context-switching tools like Software Carousel. With expanded memory, the 8088 can achieve full multitasking using windowing environments like Quarterdeck's trend-setting DESQview and, to some extent, with Microsoft's Windows.

A maximum of 32 Mb of *expanded* memory is all that can be run on any of the processors—even 486s. There has been some success with mapping LIM 4.0 EMS memory to unused address space above 65K as well, but it has been very limited. The architecture of the chip does not allow protected mode or any use beyond 1 Mb.

Until fairly recently, I would have said the lack of a protected mode would not affect most 8088 users. Today, however, the world is moving quickly to protected mode. Most users needing to run applications that require extended memory probably have access to higher-level machines—286s and 386s—anyway. Certainly, as many of the vintage 8088s expire (as one did in the middle of working on this book), they will be replaced with new machines that do support protected mode.

One of the most significant limitations of the 8088 for many applications is that the 8088 is essentially an 8-bit chip. Some call it 16 and it is—but it really isn't. The chip contains two 8-bit processors that, running side-by-side, effectively amounts to 16-bit processing. By design, the chip is limited to only 8-bit data communication with the outside world. Accordingly, the standard open-architecture bus of the PC and its various clones and spinoffs is an 8-bit bus. Given an 8-bit architecture, the operating system and any software that truly is compatible with it must be scaled to run in and through an 8-bit world.

The 8086 differs significantly from the 8088 in that it can transfer 16 bits and could, therefore, operate more effectively given access to a 16-bit bus. Internally, both chips actually can manipulate 16-bit data. The two chips, therefore, are not directly interchangeable. This, by the way, is essentially the difference between the 386SX and 386DX chips. Only the DX chips are capable of 32-bit communication (i.e., it supports a 32-bit bus), although both are 32-bit processors.

The limitations of a 1 Mb DOS and of running with an 8-bit operating system and software spurred development of a new operating system that would match the 16-bit capabilities of the 80286. These limitations led to the development of OS/2, a 16-bit operating system. In the meantime, we have the 32-bit 80386, no great enthusiasm for OS/2, which does not yet even offer a viable 16-bit alternative, a well entrenched 8-bit operating system, and highly evolved user base of 8-bit-compatible software.

Despite the dire predictions of some pundits, the 8088 is not finished yet. Even in office environments that often require the best that technology has to offer to meet certain more sophisticated needs, a lot of old PCs still are doing yeoman service. With today's software (DESQview especially), the old PCs even can multitask. With the price of 286 and even 386SX machines becoming more and more attractive, at all but the most basic entry level, you should set your sights a little higher.

The 80286

From its introduction, the 80286 has been somewhat of an orphan, caught somewhere between the PCs 8088, which it was supposed to replace, and a promise it somehow never quite fulfilled. The 80286 brought something euphemistically called *extended memory* into our jargon, along with such esoteric terms as *real* and *protected mode*. After the initial hype had died away, about all there really seemed to be was a faster PC with a different kind of RAM disk.

The 80286 could address more memory than the 8088 was capable of handling—up to 16 Mb. Specific computers using that chip, however, might have substantially lower limits imposed by other design factors. For example, the IBM AT had an official limit of 3 Mb, except for extended and/or virtual memory. Interestingly, this amount is the same limit that for many years applied to IBM's then multi-million-dollar mainframes. The 80286 also made provision for something called *protected mode*, which was supposed to assure data protection and integrity.

The idea of something like a protected mode—and several other features supported by the 80286—actually dates back to the mid-1960s and came out of a joint project sponsored by Bell Labs, MIT, and General Electric. Unfortunately, DOS had been written on the basis of some shortsighted assumptions that made no allowance for a protected mode or any possibility of life beyond 1 Mb. Therefore, as long as there was DOS and it was the only viable operating system available, there was a problem, or at least there seemed to be. While some software developers ventured cautiously into this ungoverned area called extended memory, it proved to be a risky business fraught (until rather recently) with all sorts of dangers. The result was only limited usage and a lot of bad press based largely on common misconceptions.

One of the most common of these misconceptions was the "fact" that, no matter how much extended memory you had, you could not run more than one application at a time in extended memory because the second application would most likely overwrite the first, corrupting its data, etc. You could not, for instance, use your extended memory both for a RAM disk—like VDISK—and for disk caching or print spooling simultaneously. Maintaining the integrity of several programs running simultaneously in extended memory was what protected mode was designed to prevent. Without some help from the operating system, however, you really didn't have protected mode, only the capability.

DOS still is our primary operating system; however, thanks to the development of something called *DOS extenders* the problems that plagued 286 extended memory usage have been largely overcome. In essence, a DOS extender picks up where just plain DOS leaves off and provides the software management support for extended memory that DOS lacks. Beyond 1 Mb, the extender takes on the role of surrogate for the new operating system we're still waiting for—though less and less with bated breath.

There now are two industry standards for DOS extenders and any other software running in extended memory. First, there was the Virtual Control Program Interface (VCPI), which was put together by a group that included most of the major players in the game (with the exception of Microsoft, which chose not to participate) in the interest of not shooting themselves—or each other—in the foot.

The VCPI sets down guidelines that, when followed by control programs (such as DESQview), allow the control programs to coexist without conflict in extended memory. This standard, while of critical importance, addressed only the 386 (and higher) platform level, which is where most of the interest and activity continues to be.

Unfortunately, Microsoft Windows was not compatible with the VCPI specification. Although they could have brought their products into compliance with the VCPI specification, Microsoft chose to push for a quite different interface standard and, according to various reports, threw its not inconsiderable weight around to force the new standard's acceptance.

Called the DPMI (DOS Protected Mode Interface), this specification also embraces the 80286 and, in fairness, does address some shortcomings of the earlier—though still more widely accepted and used—VCPI. The DPMI, however, does not solve all of the problems encountered in trying to use extended memory with a 286. There are some problems with the chip itself.

The basic problem with the 80286 chip—aside from the lack of an operating system or even an accepted add-on interface to protected mode until recently—is that, while the chip will transition easily from real to protected mode, there is no easy way to bring it back.

Although a program can run all day in protected mode given sufficient data to calculate, extrapolate, interpolate, and otherwise chew on, it has to come back to the real world for disk access, file management, and other DOS services. DOS ends at 1 Mb (FFFFh). Any time you go past that limit, you must go out of DOS—at least until you need DOS services again. There is one exception to this rule, which will be discussed in a later chapter. That exception, defined by Microsoft in the Extended Memory Specification Version 2.0, allows DOS to use one additional 64K lying almost entirely above 1024K.

A couple of methods are used to move back into real mode from protected mode with the 80286 or 80386. Another peculiar device, however, must be dealt with in any transiting between modes with these chips. The device, called an A20 gate, wraps calls to addresses above 1024K back around to the bottom of the address range, rather than allowing them to pass through into the extended memory range.

Apparently, at one time, a lot of programmers used the rather sloppy trick of throwing in addresses above 1024K when they really just wanted to wrap back around within DOS. This trick seems to have been a carry-over from a still earlier programming era. Unless it is specifically turned off by a command, the A20 gate will perform this "trick." Programs that really call for addresses beyond 1024K must turn this gate off before even being allowed access to extended memory. The programs also must turn it on again to re-enter real mode.

Assuming the gating instructions have all been taken care of, one trick some programmers have used to return to real mode is to take advantage of an undocumented feature of the 80286 chip discovered early on by hackers. Commonly referred to as the LOADALL function, it was likely built into the 80286 to serve some internal testing or quality control need at the factory. The function provides a relatively easy and direct means of returning from protected mode. The problem, however, is that Intel not only has never documented the function but has never given any assurance that the feature will always be there in later chips. Although the function does work, there is a definite element of risk involved for any programmer who chooses to ignore the danger signs.

The more common and more accepted solution—one that requires only well documented features of the 80286 chip—is something called a *triple fault.* Although a full technical explanation is beyond the scope of this book, the curious might find a thumbnail sketch amusing if nothing else.

When a program running in protected mode needs to return to real mode, it does something utterly outrageous and loads a register with a value that can only be interpreted as an error. Having detected the "error," the system goes looking for instructions as to how to deal with an error at that point, only to find another value indicating a second error at that level (also put there by the program). This error then sends the chip to its third and final level of error protection where it finds yet another invalid value returned.

After the third fault, the 80286 gets ready to reboot—which would clear the faults but also lose whatever you were working on. On checking its various registers, however, the chip discovers a pattern of data bits indicating that the system wasn't just turned on after all; therefore, it doesn't reboot. At least, it doesn't reboot completely, but rather just far enough to reset certain registers and to put you back into DOS. The same effect can be noted occasionally with other chips during a cold boot and somewhat more often during a warm boot. The processor chip will detect some bit pattern that indicates the system is really alive, it will only reset rather than reboot, leaving you in limbo somewhere. In those cases, a reboot must be forced, typically by shutting down the power to force a total restart.

Convoluted and weird? You bet. But dependable enough. It also is the method of choice of most 286 programmers. Until the doors to extended memory were opened wide by the 80386, however, there was little incentive to bother. Besides, OS/2 was coming and the problem would just go away. The problem, however, didn't go away. In the meantime, the 80386 is here and blowing the doors off of everything—hence the sudden scramble to catch the wave.

The 80286, however, has some serious shortcomings, which are made all the more apparent in the glare of the spotlight on the 386. Unlike the 386, the address registers of the 286 cannot be remapped to allow tucking bits and bigger chunks of usable RAM into unused and otherwise unoccupied address spaces between 640K and DOS's 1 Mb top. The 286 also does not have the virtual machine multitasking capability inherent in the 386. Some users even go so far as to call the 286 a "brain dead" chip because of all the things it can't do.

There is, however, a large installed user base of 286 machines out there that is getting bigger every day. People are still buying 286 machines often because they simply do not think they can justify the added initial cost. In addition, many users simply do not now and might never need the performance achievable with a 386. It would be as silly to buy a 386 just to balance the family budget or write high school term papers as to buy a high-performance sports car only to drive to the corner market occasionally for a loaf of bread.

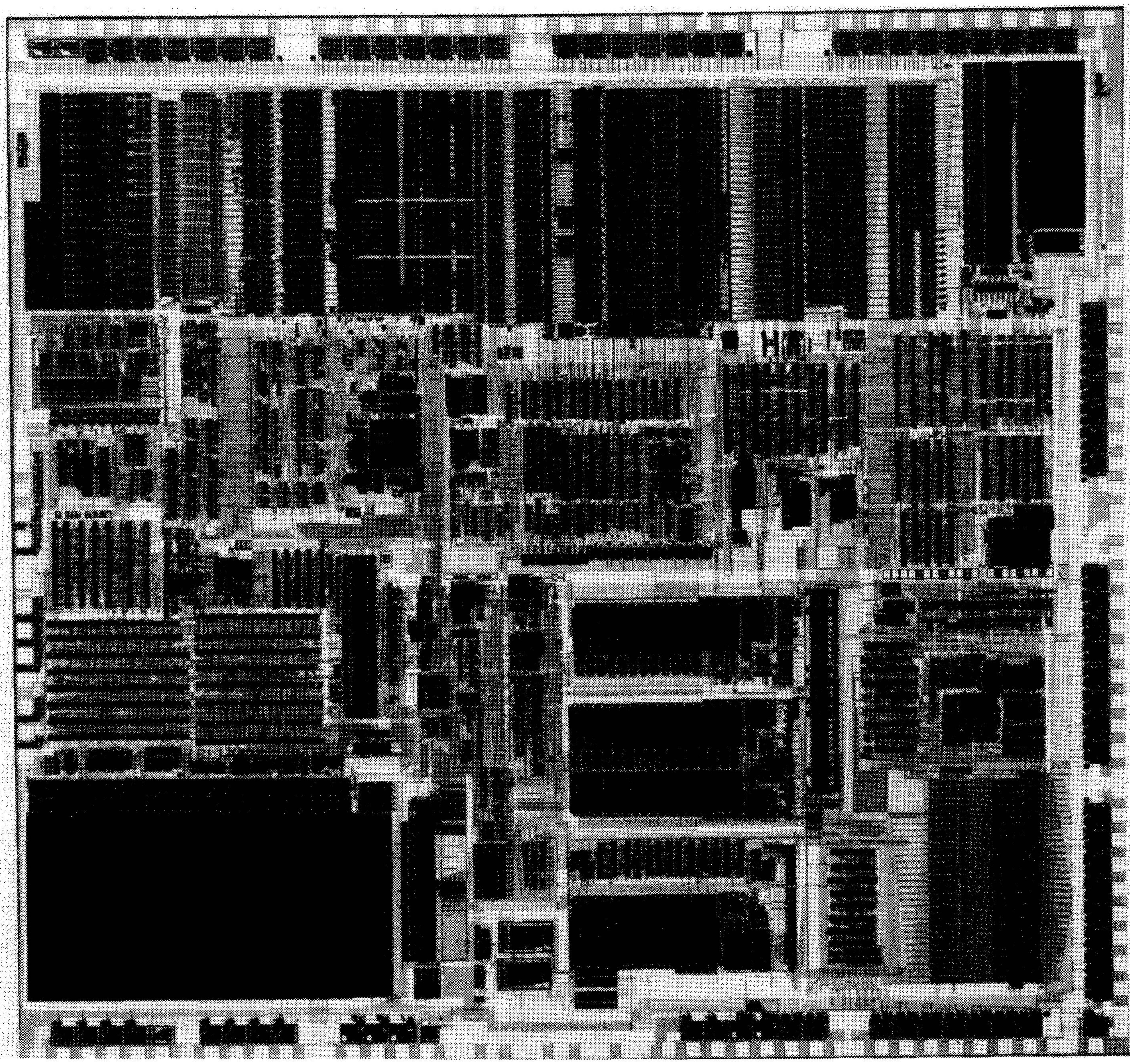
Needs do change though—sometimes as new technologies make new applications available and access to them desirable if not imperative. Fortunately, there are solutions at hand that address most of these shortcomings. In this case, the real solution requires some additional hardware to provide services not supplied by the raw 286 chip. As with the 8088, there is always the accelerator board upgrade option, which will be discussed in a later chapter.

The supercharged 80386

The original 80386—80386DX to be precise—and its closest relative, the slightly scaled-down 80386SX, have emerged the biggest winners in the technology sweepstakes. Now, yet a third jewel is added to this crown: the 386SL, scaled down in size, but hardly in performance. To date, this family of chips has surely done more to revolutionize the desktop computer and its role in our lives than any other single development. Even with what we've seen so far, the 80386 (Fig. 2-2) is still a sleeping giant just awakening.

Some users, unfortunately, still look upon all 386 computers as little more than Advanced Technology (AT) warmed over, but with a fancier price tag. In some cases, that view is accurate. The AT image has been fostered further by the fact that, lacking a classification of their own, 386s are often classed as ATs in generic listings that put PCs on one side and everything else on the other.

There are similarities of course—particularly between SX and 286 machines. If you had to classify the 386 either with the PCs or ATs the latter is certainly the



2-2 The internal structure of an Intel 80386 is just one indication of the incredible complexity of modern microprocessor chips. Just imagine what an 80486 might look like, with over 1,000,000 transistors.

better choice. However, 386s—even 386 machines that look for all the world like old ATs—are indeed a different breed. Throughout this book I will treat them as such.

As stated up front, I will for the most part refrain from using the terms PC or AT in this book, but will group machines strictly according to the microprocessor chip they are based on. Again, I would remind you that even that classification is risky business because all 386s are not created equal by any means, so all of the features described here might not be available on all machines you see advertised on dealer's shelves.

The real and the unreal

After the 286 fiasco, Intel made sure everything had been done right when it introduced the 80386. It was not just a fix—the 286 chip really seems to be unfixable—the 80386 was a whole new chip. It was and is compatible with anything that runs on any member of 8086 dynasty. The 80386 promised higher clock speeds than had been seen with 286s. The real news, however, was in features that had never before been seen.

One new feature allowed memory to be mapped to any block of addresses no matter where (within an acceptable range) the addresses were. Although this feature might not sound like much, it was, in fact, a revolution in the making and a probably whole new lease on life for DOS. For the first time, user memory could be assigned to unused blocks of address space up in the system area above 640K. The memory could be worked into little unused nooks and crannies up above the video area and worked in around such things as network cards, hard disk ROM, the page frame for expanded memory, and system ROM.

This RAM could not be used directly to run bigger applications—DOS can run applications only in contiguous memory, which ends when you hit the block of addresses used by the video. Further complicating the situation, the unused address space above 640K usually is fragmented (8K here, 32K there, 64 or 96K somewhere else). Unused memory in the system area, however, could be used to relocate device drivers and TSRs that were using precious memory below 640K. Every program moved to higher memory makes that much memory available below 640K—up to nearly 200K sometimes.

Certainly one of the most significant—although probably least understood—differences in the 386 chips, however, is that they actually have three distinct operating modes. In real mode, a 386 machine is really just a big, lovable PC. It looks impressive on anyone's desk—especially as most other new machines take on a trendy, slimmed-down look and most 386s run at real "gee whiz" speeds by comparison. At heart, 386 machines are still just big PCs, their 32-bit processing capabilities wasted.

Shift into protected mode, however, and the differences are quite apparent. Unlike the 286, the 386 can be readily brought back from protected to real mode at will, without the convoluted skullduggery required to sneak back when using a 286. Some implementations of OS/2 have been able to use this feature effectively to allow real-mode MS-DOS programs to run concurrently with OS/2 applications running in protected mode.

Even without going to OS/2 or some yet-to-emerge operating system, protected mode really has come into its own with the 80386. Programs run from DOS—not some new, exotic, and totally unfamiliar operating system—can load and run in extended memory and take full advantage not only of an utterly mind-boggling pool of memory, but also of the full 32-bit processing capabilities of the chip. Translated to demonstrated performance against the same products fine-tuned for use in a 16-bit environment, the results can reflect an increase by a factor of five or more. Even under DOS, this processing power is unleashed, thanks to DOS extenders.

True, DOS extender technology is being applied to software specifically targeted for 16-bit 80286 systems as well. However, much of the current DOS extender technology being applied to 286s could be said to be more hand-me-downs from 386 development rather than from dedicated efforts. Although there are strong similarities, DOS extenders for use with the 80386 differ significantly from those intended for use with the 80286. Some of the better known specialized software development companies—like Phar Lap—only offer development tools for 32-bit systems, totally ignoring the 286 market. The 80386, however, is clearly the more attractive development platform. The development of 32-bit system-specific versions of several popular software packages seems far more likely than parallel 16-bit releases. There also is a rapidly expanding base of other software developed specifically for the 32-bit environment. Much of this software probably will not filter down even one notch to the 286 market.

What has been discussed so far is only part of the 386 story, for the 386 has still another operating mode yet to explore. This mode called *virtual 8086* mode, is not shared with any of the earlier chips. While opening the doors to new power, the 386 opens windows like no other chip can. In this V86 mode, the chip can emulate virtually an unlimited number of separate 8086 machines running side by side, each with its own operating system and configuration—each one actually running real mode software in a protected mode.

The virtual machine

The emulation of separate virtual machines is so complete that the user can run different virtual machines under different operating systems—different versions of DOS or even other operating systems—provided they are compatible with the 8086 family of processors. Virtual machines also do multitasking like you've never seen it done before. Running in real mode, we've had multitasking now for several years even on 8088s with LIM EMS 4.0 expanded memory. This real mode multitasking is different though because, in real mode, a problem encountered by one multitasking application can bring the entire system down. Fortunately, it usually doesn't, but it can.

With full-fledged virtual machines, however, the separation is complete, even to the extent that, if you totally crash a virtual machine, whatever other virtual machines you might be running at the moment never know it. You simply reboot the one that crashed, and you're right back in business. The result is about the same as having a bunch of physical machines piled on your desk and having one of them crash and be rebooted.

You still can't get around the fact that, in real or virtual mode alike, you have only one processor that has to share itself between the different applications that you have running concurrently. In any event, however, this virtual machine mode opens more than just new windows; it also opens doors.

Big daddy

The "big daddy" of the family, the full 80386DX, is the chip now emerging as probably the most cost-effective network server. Even though we already have the still more powerful *i*486, there is a growing consensus that the 80386DX has all the horsepower needed for all but the most demanding uses.

The DX has more address pins than any previous chip—32 of them—giving it the capability of addressing up to 4 gigabytes of memory. Two versions of the DX are available that support clock speeds that are appreciably faster than can be expected out of even some of the faster 286s. These versions are the two you hear bandied about the most; however, neither of them really warrants the added cost over a 286 to most users. To justify the cost, you have to look a little deeper.

Although the DX was the first 386 we saw, it has only more recently emerged to show its real power (partly because many of the early 386 machines were in fact little more than warmed-over AT clones). It also took a couple of years for software written specifically to tap the power of this chip to reach the market in sufficient quantity to make a real impression.

Only with the teaming of the 80386 with more powerful full 32-bit EISA bus and finally the Micro Channel architecture as well did the 80386 begin to really show its muscle. The power of this chip, however, still had not been fully unleashed. Modifications to the basic EISA bus now have extended the data path to as much as 128 bits on some machines. Such wider data paths are essential as designers talk in data transfer rates as high as 10 Mb per second these days.

The 386DX is an awesome chip and should not be sold short, even along side the new darling of the industry, the *i*486. No one as yet has even approached the DXs 4 gigabyte address limits. Buyers of 80386 machines in general and DXs in particular, however, should be wary. As powerful as it is, the 80386DX in itself is only just a chip. Only when it is incorporated in an overall system designed to fully utilize its powers can you enjoy the DX's power, otherwise, you might be better off buying a lesser machine and save yourself some money.

The entry-level 386SX

Similar internally to a full 386, the SX (Fig. 2-3) opens the doors to full 32-bit processing, running any of the new 32-bit software, including 386 DOS-extended software that will not run on 286s. Because they are descended from the 386, the SX offers the same three operating modes, the same support and easy access to Extended Memory (no backflips to get in and out of protected mode), and the same support for memory mapping to unused address space above 640K.

Essentially, the 386SX can do anything a full-blown 386 can do except address 4 gigabytes. It has fewer address pins (only 24 instead of 32). The chip

A80386-20
© i '85 '86
intel
80386SX-16
© INTEL '85 '87

Intel

2-3 Significantly smaller and different in design from the original 80386, the 80386SX offers many of the same benefits, but at a lower cost to systems developers. While only nominally faster than the 80286 and with a smaller address range, it can be used with virtually all 32-bit software written for the 80386.

still can handle up to 16 Mb of RAM—certainly enough to satisfy most user needs.

Although the SX performs 32-bit processing internally, it does not support a 32-bit bus. It supports only a 16-bit bus, the same as the old 286 AT. This apparent limitation, however, makes the SX more competitive, because producing a 16-bit-bus motherboard is significantly less costly than producing the 32-bit kind. The net result is a machine that will run anything a full 386 can run, including many programs that can't run on 286, at a price generally significantly lower than that of a comparable DX machine. All other things being equal, an SX typically sells for only $100 to $200 more than a 286 machine. Dollar for dollar, a 386SX is, without a doubt, the best buy on the market today for most user applications and should continue to be for some time to come.

Power to go: the 386SL

It is anticipated that by 1994 as much as 37 percent of a projected 40 million-unit PC market will be for laptops or the smaller notebook sizes. In the same timeframe, 32-bit systems could account for more than half of all the portable machines. Looking to that market, Intel now has added the SL to its 386 line. The 386SL is a diminutive chip that, along with a matched set of preshrunken components—also offered by Intel—fits nicely on a board as small as 4×6 inches.

This tiny powerhouse might be small in size, but not at the expense of major features. Internally, the diminutive SL contains the equivalent of no less than 885,000 transistors. In addition to the core, the SL contains its own internal clock, memory, and ISA bus-compatible controllers. It also contains its own SRAM cache and cache controller.

More than just shrinking the physical size, Intel also has devoted considerable effort into reducing battery drain and has incorporated power management capabilities, such as stopping the clock during periods of inactivity to achieve the longest battery-powered work session possible between trips to the recharger.

The SL already is available at OEM prices that seem surprisingly attractive (under $200) considering the relatively higher price-per-horsepower tags we've seen to date on anything laying serious claim to portability. The area of portable computers seems likely to be the market where we can expect to see not only the biggest quantum leap in performance but also the most significant price reductions, bringing those prices more into line with the bulky desktop machines that will offer little, if any, added power.

Above and beyond: the *i*486

With the 386 by no means the end of the line, the current darling of the industry is the *i*486. Outwardly little more than a souped-up 386, internally it represents another quantum leap in CPU technology. In the near future, however, the 486 apparently will have little impact on the way most of us live and work.

The *i*486 is a far superior number cruncher to the 386, with internals that essentially make it equivalent to running a 386 with a math coprocessor. If you've priced Weitek or other better coprocessors for 386s of late, the price would go a good way toward offsetting the significantly higher *i*486 price tag. To really put the 486 in a class all by itself, it also supports the use of a coprocessor such as the Weitek 4167.

The *i*486 has other attributes that many users feel make it especially well-suited to network server applications. It really is too much chip for the AT-type ISA bus—even modified AT bus boards like we've seen in many 386s that provide at least limited 32-bit access. For that reason, most *i*486 machines are being designed around the EISA bus, or Micro Channel Architecture at the very least. Some engineers are even modifying the basic EISA design to increase the data path to as much as 128 bits.

Unlike the 286s and 386s, the *i*486 does not add any new or novel features. It is still a tri-mode, 32-bit chip. It will not spawn a whole new genre of 486-specific software. In the context of this book above 640K it looks little different from a 386 above 640K.

An SX version of the *i*486

In a move that so far has raised more eyebrows than excitement, Intel scaled-down its 486DX and made an SX out of it. What Intel has done, essentially, is to eliminate the internal math coprocessor, which was the key to the incredible number-crunching capabilities of the *i*486DX and one of the features that most distinguished it from the 80386DX.

If number crunching is what you need, Intel will sell you a 487 math coprocessor, which Intel claims will crunch numbers up to 40 percent faster than a coprocessor-equipped 386. At this time, however, Intel's price for the coprocessor alone is more than three times the going price for the 486SX itself—not the kind of numbers that are likely to set off a stampede.

The 486SX does seem to have some advantages over a 386DX, but the 486SX certainly does not represent the kind of quantum leap in CPU technology the full 486DX demonstrates. The price of the *i*486SX is expected to remain somewhat higher than that of an 80386DX, at least in the foreseeable future. Given this, it has been widely speculated that, rather than trying to fill a market void, the motivation behind the development of a neutered 486 has been more a matter of trying to preserve market share in the face of the increasingly aggressive marketing of cloned 386 chips by at least two other vendors at this writing.

Boosting performance even more . . . sometimes

The performance of any processor from the 8088 on up can be enhanced still more by the addition of a matched coprocessor chip. Generally, these chips are numbered the same as the main CPU with the exception of the last digit that, in the case of matched coprocessors, is a 7 (8087, 80287, 80387, and etc.).

The role of these coprocessors, however, is misunderstood by many users. Installing a coprocessor does not automatically increase the processing power of your CPU. Coprocessors (often referred to as math coprocessors) are exactly that: math coprocessors. They crunch numbers. Although everything computers do involves some form of number crunching, coprocessors are very selective in what numbers they will crunch. The bottom line is that, even if you're doing spreadsheets, for instance, it is quite likely you will see no improvement at all. You will certainly see nothing to justify the investment that, in the case of something like one of the Weitek coprocessors for 386s, can run to $1000 or more. Not that there are not many applications that will benefit from the use of a coprocessor. Many will. There are even software packages, such as some releases of AutoCAD, that will not even run unless you have one. The best advice, however, would be not to buy a coprocessor unless the software you are or expect to be running specifically requires or, at least, strongly recommends the use of one. Otherwise, you're just wasting your money.

While I'm on the subject of coprocessors, there has been some interest of late in the *i*860 chip used as a coprocessor in some high-end situations. The performance figures are sometimes spectacular: 4 to 5 times, even to as much as 11 times faster than the unassisted 486, which already has the equivalent of a math coprocessor built into the basic chip itself. Don't hold your breath, however, waiting to start seeing spinoff benefits.

The *i*860 is a RISC chip (Reduced Instruction Set Computing), which means it is not compatible with the DOS instruction set but rather must be given its instructions separately from those passed to the primary CPU chip. Unlike conventional math coprocessors that can take their instructions right along with the main CPU, the software has to generate a special set of RISC instructions, too. You could not, for example, take a program like AutoCAD that, as mentioned before, in certain releases requires a coprocessor and expect the program to happily embrace an *i*860. We're talking special software here—software written specifically to feed the *i*860 with the instruction set it needs.

The *i*860 points to what seems to be a growing trend these days to the use of more than just a single CPU chip in machines. Chip design in many ways has reached the outer limits—at least the outer limits as we know them with today's technology. Barring some major breakthrough, we have long since reached clock speeds that, in many cases, cannot be supported by the rest of the system. Wait states, interleaving, and other schemes are ample evidence that even the best RAM chips available can't keep up.

A ticking clock

No matter what we do, we always seem to come back to clock speed. Clock speeds have grown by leaps and bounds from 4.77 MHz to a demonstrated speed of no less than 100 MHz. Most of us live at speeds well above those old PCs these days, but generally at something less than half of the speed Intel has demonstrated—with a chip literally running so hot that it required internal cooling.

Clock speeds are probably one of the most deceptive measures of computer performance. Even at the more modest middle speeds most of us run at these days, systems are pushed right to the limit or beyond, by inserting something called *wait states*, which are pauses that allow the rest of the system time to catch up, not just occasionally but on a regular basis. *Wait states* can, and often do, slow the effective system speed down to half of the clock speed, sometimes even less.

One of the most severely limiting factors is the kind of RAM we use—not just the rated speed of the chips (which at best is no match for today's clock speeds) but the physical nature of the chips themselves. Most machines today use something called DRAM, or Dynamic RAM chips. By resorting to various design tricks like interleaving banks of DRAM chips, manufacturers can mask how slow these chips really are, up to a point. Beyond that point lie wait states, which means you're not getting the performance you thought you paid for.

To genuinely increase the speed of the overall system, we have to get away from the use of DRAM and use a faster media. SRAM—Static RAM—chips fall into that category, with refresh rates typically as short as 25 nanoseconds. Running a machine to about 50 MHz without any wait states should be possible using SRAM; however, these chips are much more expensive and few manufacturers have shown much enthusiasm yet. CompuAdd, one of the few enthusiastic manufacturers offers a SRAM daughter board as an option for one of its high-end tower machines with enough SRAM to cover the needs of the DOS area.

Another factor that severely limits system performance in today's increasingly popular multitasking environment is something called *time slicing*. You take a chunk of time and slice it into ticks of the computer's clock: two ticks for application A, one tick for B, etc. In this example, during two ticks, A can have the microprocessor, but then it has to stand aside for B to have a turn, and so forth.

In the example, in which I have divided things into a 2/1 split, the effective clock speed of our primary application is only $^2/_3$ the actual clock speed, our secondary application only $^1/_3$. A 16 MHz 386 system then is only showing that second application a little more than 5 MHz. The old, original 8088 PC ran at 4.77 MHz, and everybody cried, "It's too slow!"

If you divide the available ticks among more than two applications, the situation gets even worse. The system slows down fast, if you'll pardon the seeming contradiction of the statement. How you allocate the ticks is an option you, the user, usually has a say in. You can be stingy or generous depending on your individual needs.

If you've got a lot of numbers that you've got to crunch and have all day to work on them just so long as you can print out a report by five o'clock, put that program on a slow back burner. Let it simmer in the background quietly until it's done. In the meantime you've still got the lion's share of the clock ticks left to run another application in the foreground—and maybe yet another in the background.

Clock ticks, however, are a resource like megabytes, monitors, and everything else that has some finite value. Unless you are involved with multitasking, you might have never noticed. "Okay," you say, "so everything just runs a little slower. It's no big deal. I can live with that . . . I think." You are going to have to. Some of your software, however, might not be happy with the arrangement unless you allocate more clock ticks than you'd really like to (high-speed data communications running in the background, for example).

Although many of the new 386s run at clock speeds that, at a glance, might seem absurd, remember that every tick of the computer's clock is a finite resource that only can allow a program time enough to do a certain finite measure of work. If you've never done multitasking, it might not seem like such a big deal. Once you get into multitasking and have to start handing out the ticks, it will become a big deal.

Fortunately, there are ways to get around the problem. You can't get around it completely but you at least can minimize it. Background programs can be polled, for instance. If they are not actually running code or performing any I/O, their time slice can be shortened. Some schemes even skip past applications—even in the foreground—that have been completely idle for a time or two to devote more time to something else that needs that time. A word processor in the foreground, for instance, might be waiting for the next keystroke for half an hour, while dBASE is trying to sort an index and needing all the time it can get.

No matter how you slice it, time is a finite resource. You can juggle, you can share it. You can do almost anything with it, except stretch it. With multitasking and a little help, you can almost seem to stretch time sometimes.

There's more

There are a number of other hardware issues surrounding the underlying system. I have tried to limit the discussion here to just those especially important issues, as we venture ever farther into the areas of extended and expanded memory and as our daily applications force us to press on still farther.

Particularly important are the various ways manufacturers install and implement whatever system memory you start with—extended, expanded, or conventional. These and other questions pertaining to memory and memory utilization become more significant when considered against the broader picture of memory usage and management.

3
CHAPTER

The new breed

Unfortunately, no matter what kind of microprocessor chip you're using, the box it comes in looks pretty much the same. A box is a box, and this year's box looks pretty much like last year's or the year before's. They all have a bunch of memory stuffed in somewhere. If you know how many Ks you have, that's all you have to know about the memory, right?

Wrong! That statement might have been true at one time, but not any more, especially now with all the advancements in extended and particularly expanded memory technology. True, the primary subject of this book is what's happening beyond the barriers of DOS; however, conventional memory and the way it is accessed other hardware provides—or should provide—a foundation on which to build. Unfortunately, some of those foundations are pretty shaky.

A brave new world

The situation, already chaotic, has become further muddied with the advent of the DOS extender and the belated recognition that the DOS-based world can no longer be bound by the restrictive limits of the 8088. With that recognition, total and absolute backward compatibility, one of the unshakable cornerstones of DOS, began to crumble.

DOS, beginning with version 5.0 especially, now caters to a split-level hardware base. It is and always will remain compatible with 8086/8088 machines. Even DOS now offers advanced support features applicable only to 80286 and higher machines and, in the case of expanded memory emulation, only to the 80386 and up. Much of this support is self-serving, geared to the needs of Windows 3.0. The mere fact of 3.0's existence, however, is a recognition of a brave new world beyond DOS's traditional 1 Mb that is accessible from DOS.

Microsoft did not spearhead this new wave, rather it was simply caught up in it. The wave was generated by a lot of other players. Names like Quarterdeck and Phar Lap have played a major role. They were movers and shakers, if you will. This wave has resulted in incremental levels of device-specific software—some requiring a 286 or higher, but more commonly at least a 386. At the same time, manufacturers are rethinking not only the mission of the machine but also the philosophical approach to its achievement through design.

What Microsoft's active participation has done more than anything else is to raise a flag—a rallying point—from which the industry can go out, supported by the operating system rather than working around it. Some issues, which were paramount before, now must be reexamined in the light of different hardware platforms and performance levels. Other new issues, however, just now are emerging.

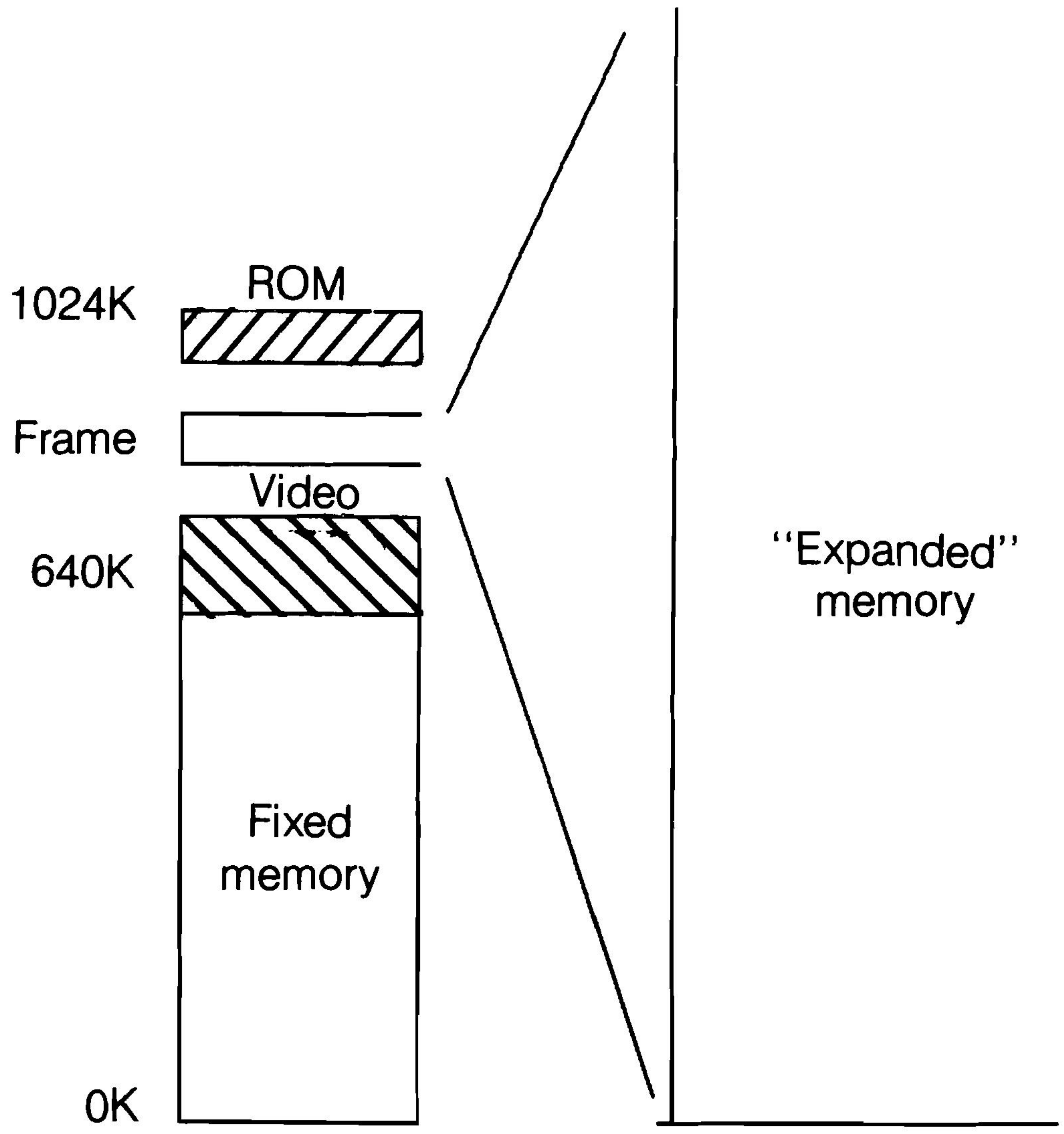
We can no longer deal with hardware or software in generic terms, where anything that is compatible with DOS can run on any DOS machine. Although much of what is covered in this book applies to any DOS machine, in many places I will be looking at specific categories based mainly on the strengths or weaknesses of the CPU chips they incorporate. I, however, need to establish some ground rules before going on.

Games anyone can play

Expanded memory, the first glimpse most of us had of life beyond 640K, is and will remain the only way available (even to 8088 users) to access large amounts of memory. Born of desperation, Lotus was one of the driving forces behind expanded memory's development as a means of satisfying the needs of Lotus users for more memory to support bigger spreadsheets. No longer the darling it was when it was the only game in town, expanded memory nonetheless played a major role in the evolution of computers as we know them today.

Expanded memory, however, isn't going to fade away into retirement. Expanded memory not only does, and will continue to, greatly enhance the capabilities of the firmly entrenched 8088 user base, but also offers all the added power or convenience needed by a lot of software. As long as there are 8088s in use in substantial numbers, software developers will, wherever practical, look to expanded memory usage first to appeal to the broadest possible market.

The beauty of expanded memory is that any DOS machine can use it—an original 8088 PC or a shiny new *i*486 alike. The 8088 was the only chip available when expanded memory came on the scene. The key is that expanded memory uses addresses within DOS's nominal 1 Mb limits as a window through which, with a little sleight-of-hand—small (16K) blocks, or *pages*, of memory lying outside that megabyte can be accessed. This window (Fig. 3-1), typically located above the video area but below the ROM area, is called a *page frame*. A page frame generally accommodates four 16K pages at a time, swapping the actual blocks of memory the pages represent in and out of the window as required.

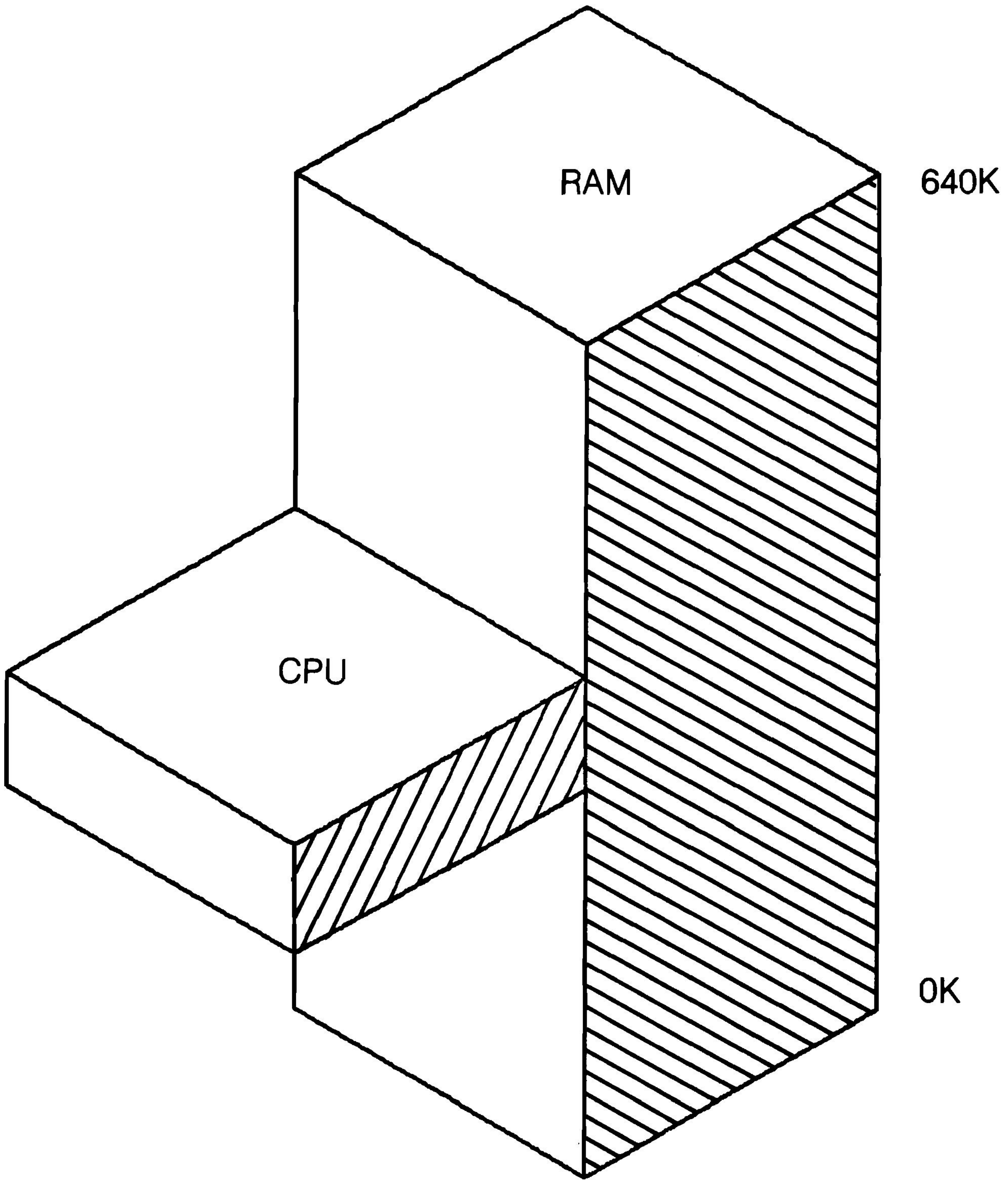


3-1 This figure represents a traditional pre-EEMS/LIM 4.0 EMS system with both Fixed Conventional Memory and Expanded Memory as first implemented under the original LIM specification.

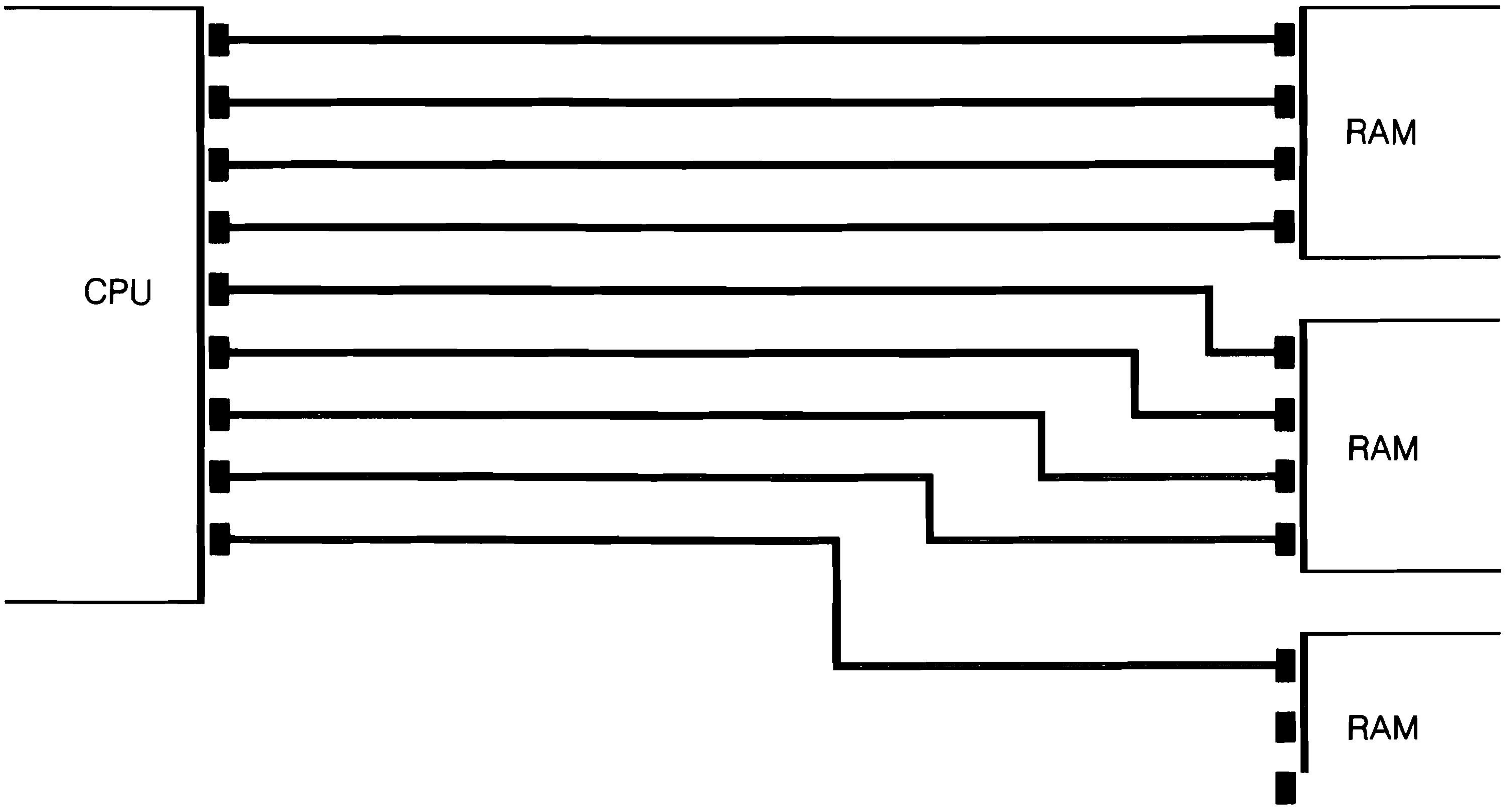
Borrowing on technology IBM had long been using with mainframes, there was nothing really new or innovative about bringing expanded memory to the then fledgling, memory-starved 8088 machines—even Apple was doing it. I, however, did not say DOS was doing the swapping. Prior to version 5.0, DOS pretty much just turned up its nose and looked the other way, treating calls to the addresses below 1024K used by programs that called for expanded memory just like calls to any other legitimate DOS addresses. I'll show how the sleight-of-hand is done later on.

Although any DOS machine can support up to 32 Mb of expanded memory, the way that support is implemented varies greatly, both in the design of the underlying machines themselves and in the design of expansion products. Interestingly, those differences are not so much a function of the CPU but rather more the philosophical (loosely translated: cost-cutting) approach taken by the manufacturer. In practice, however, full support for all of the features codified by the current LIM EMS specification (4.0) is more often lacking—or more noticeable at least—in 8088 machines than in machines offering more memory use options. The problem is in how we access, or manage, the memory resources we have available.

In the good old days when the PC was king, memory management did not exist. Memory was something you either had or didn't have. The traditional relationship between the memory you had and the CPU was pretty much as shown in Fig. 3-2. Viewed at board level inside such a machine, this relationship would be even more apparent. If you took the trouble to follow the little foil traces from point to point, you would actually find specific pins on specific RAM chips hardwired to specific address pins on the system CPU (Fig. 3-3).



3-2 As originally conceived, whatever RAM was installed below 640K (704K to as much as 736K with some pre-EGA systems) was essentially hardwired to the CPU.



3-3 In early PCs, specific CPU address pins were hardwired to specific RAM chip address pins or socketing that could support chips answering calls to those specific addresses only. Similar schemes, still found to varying degrees in newer computers, severely limit memory options.

There never was any question about where a call to any specific memory address went. Whether the chip was on the motherboard or on an expansion board did not matter. You could have traced directly from the address pins on the CPU chip right to a particular RAM chip, like following a road map from the tax collector's office right to your front door. Houses don't move (usually) and neither does memory.

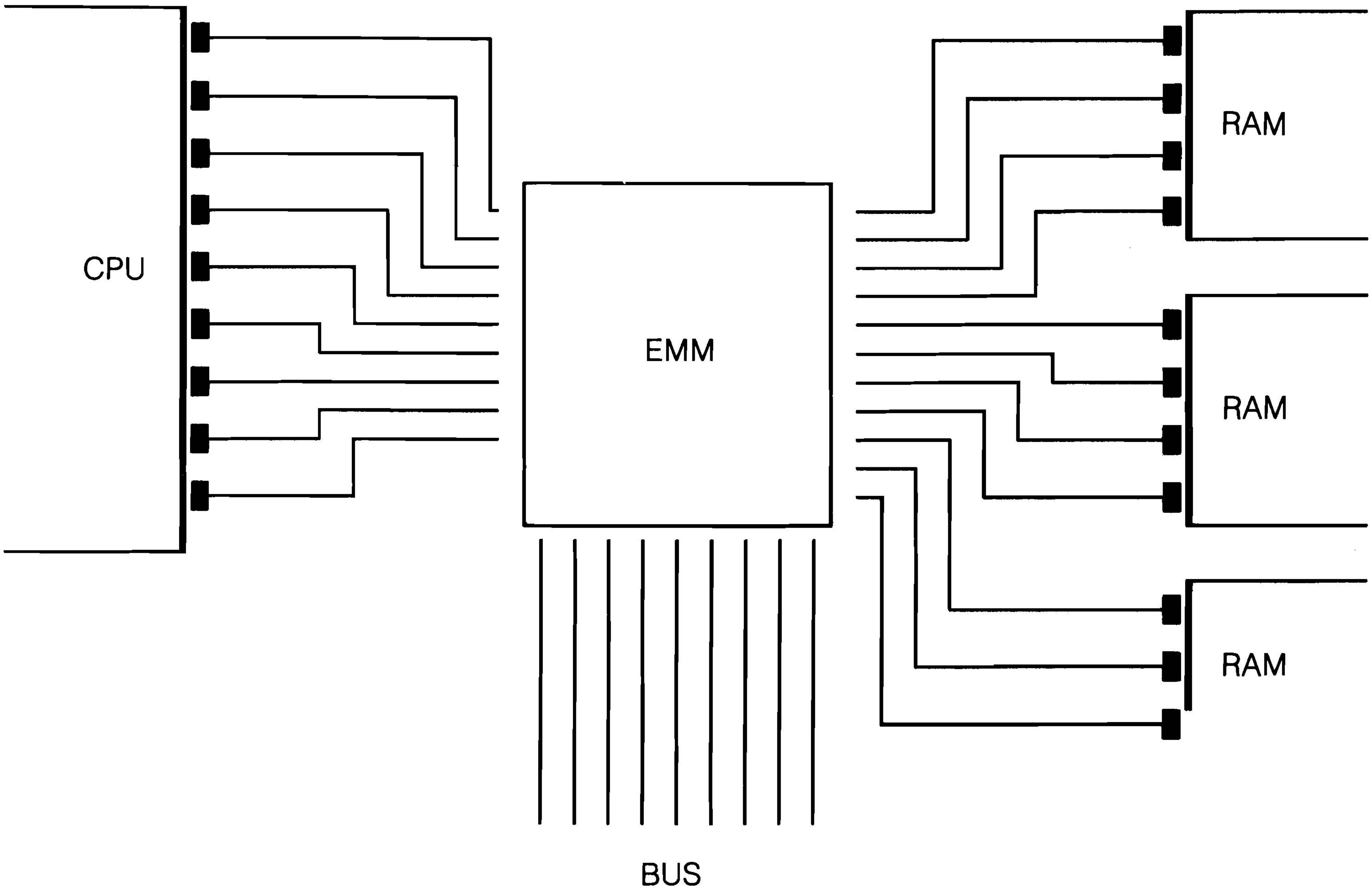
Although the advent of expanded memory did allow tasks to be swapped back and forth between conventional and expanded memory, hardwired memory did not allow concurrent processing except on a very limited basis. Multitasking was limited essentially to how many programs you could load and run simultaneously in conventional memory. Code and data swapped to expanded memory essentially went into limbo, offering little advantage over swapping them to disk (except for speed).

There were those who argued that the LIM EMS 3.2 specification—then the industry standard and the one that at least got the ball rolling—did not go far enough. Most of their argument fell on deaf ears at the time. One of the struggling innovators of the day, AST Research—then known only for its line of memory expansion boards—decided to take matters into its own hands and put its money where its mouth was.

AST Research could do nothing about the hardwired memory on the motherboard. At the time, however, many motherboards—the IBM PC included—would accommodate a maximum of only 256K. Any more than that amount required an expansion card. The better expansion cards of the day would fill any gaps up to 640K, then assign any leftover memory as expanded memory. As long as it didn't upset anybody's applecart, AST pretty much had free rein with the memory above whatever point the motherboard memory stopped. Unlike most of its competition, AST didn't hardwire any of the memory, even the part that would be used to bring the basic system memory up to a full 640K.

The AST Rampage line differed from its predecessors and its competition by putting all of its onboard memory at the disposal of a memory manager. It also let the memory manager decide, according to the user's wishes, what memory went where. Any CPU calls to addresses beyond those filled on the motherboard were intercepted and rerouted to some block of physical RAM, as depicted in Fig. 3-4.

By allowing the memory manager to intercede and arbitrate all calls for memory beyond what is on the motherboard, the Rampage also allowed the memory manager to switch the entire block it had loaned the CPU for a different block. The manager could swap the entire contents of these two (or more) blocks—code and data—in and out, not in the several seconds it took with Software Carousel running under LIM EMS 3.2 rules, but could swap almost instantaneously. With Rampage, the manager could make several such swaps a second. AST wrote a superset for the EMS 3.2 specification called the EEMS (Enhanced Expanded Memory Specification).



3-4 In contrast to Fig. 3-3, most well-designed computers today tie only some small part of installed memory (generally no more than 256K) to fixed addresses. The remainder, as shown here, looks to a software memory manager for the allocation that best suits its immediate needs. This allows much more flexibility and more efficient usage.

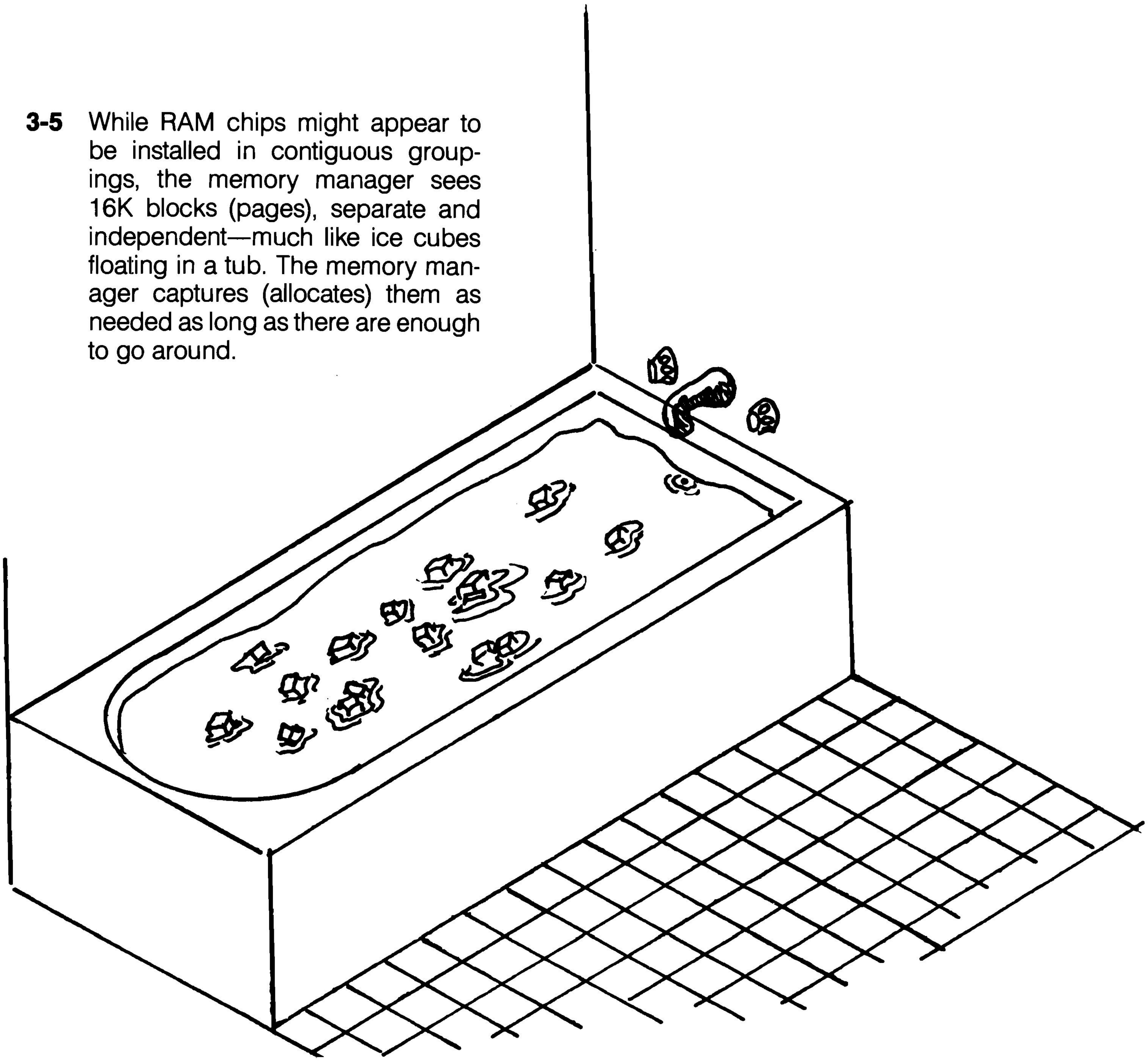
The revolution almost no one noticed

At the time, no one, not even AST, had any really good idea what all of this memory managing was good for. Lotus, which had pushed the expanded memory in the first place, was quite happy with the 3.2 specification. The company never even fully utilitized the features the specifications provided for. Others, including hardware manufacturers and software developers alike, also were singularly unimpressed.

If not for another little, then almost unknown company called Quarterdeck Office Systems, the idea might have died. It saw in the Rampage scheme the key to multitasking: swapping applications in and out so rapidly they would appear to be actually running concurrently, although only one could actually have the CPU's attention at any given time. The memory manager, with a little encouragement, just kept swapping, while the CPU, oblivious to any of it, just kept simplemindedly processing whatever task it saw.

To accomplish this swapping, the memory manager had to be extremely powerful. It had to have the power to allocate, deallocate, and reallocate the memory resources it commanded freely, reusing scattered blocks formerly assigned to exited applications in much the same way DOS reallocated disk space, often fragmenting files in the process. Blocks of memory became free agents, like ice cubes floating freely in a bathtub (Fig. 3-5).

AST would not market a computer under its name for several years yet, but the revolution had begun. Unfortunately, even several years after the merit and many benefits of the scheme had been proved beyond any doubt and considerable software had been written to take advantage of these features, many manufacturers would continue building hardware the old hardwired way. These manufacturers included not only most expansion products manufacturers but even such notable names as IBM.



3-5 While RAM chips might appear to be installed in contiguous groupings, the memory manager sees 16K blocks (pages), separate and independent—much like ice cubes floating in a tub. The memory manager captures (allocates) them as needed as long as there are enough to go around.

Meanwhile back at the ranch

At this point, I need to backtrack just a little, because expanded memory at best was only just a stopgap, not a permanent solution. Intel had developed a new chip that would be the successor to the PC's 8088: the 80286. Widely hailed as the promise for the future, the 80286 was first utilized in IBM's AT (Advanced Technology) machine. Besides being a speed demon by PC standards, the AT brought with it *extended memory*—linear memory with addresses beginning where DOS's megabyte of addresses left off and continuing to as much as 12 Mb. (The 80286 chip could address up to 16 Mb of memory but, as originally implemented at the AT, was limited by the design of the machine to 12 Mb.)

Unlike 8088 machines, the AT and the many clones that followed incorporated an 8-bit-compatible bus that also would accommodate a new generation of 16-bit hardware to take advantage of the greatly increased power of new chip. DOS, essentially an 8-bit system, was obviously badly outclassed by this new machine. A new operating system, however, was already in the works: OS/2.

The new extended memory feature was interesting; however, because beyond DOS's address limits, about all extended memory was good for was a RAM disk. The new DOS 3.0 that was introduced with the AT provided a utility called VDISK.SYS that could create a RAM disk in extended memory. Extended memory, however, promised a great deal more than that, including not only up to several megabytes of user memory but something called *protected mode* to keep applications that were running in extended memory from interfering with each other.

There was only one problem. The 80286 chip didn't work the way it was supposed to. Intel had goofed. The chip allowed software to cross over into protected mode; however, getting back into real mode for such services as keyboard input, disk reads and writes, etc., was almost impossible. This problem was a design defect that could not be corrected, making the AT little more than a faster PC that could support a VDISK (virtual disk) but little else in extended memory.

There were a lot of red faces over the 286 fiasco. The promised new operating system did not materialize. Even if it had appeared, the operating system could not have overcome the problems inherent in the chip. Extended memory drifted into sort of a technological limbo. A few software developers cast hungry eyes on extended memory. Some even used it; however, lacking any generally accepted set of rules, they often used it in conflicting ways with catastrophic results that gave the whole thing a bad name.

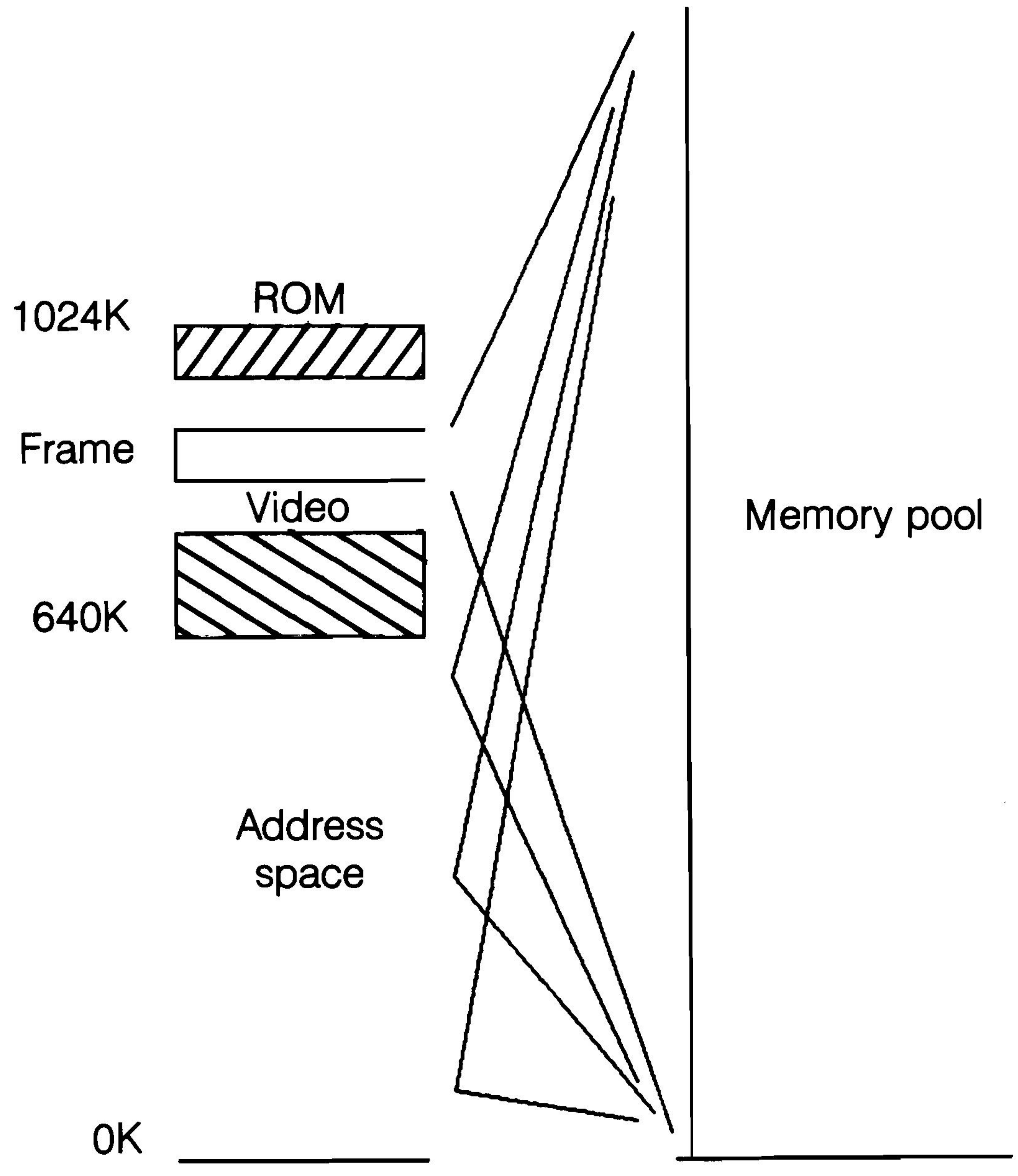
To the beat of a different drummer

In the meantime, AST's EEMS scheme drew increasing attention. Thanks mainly to Quarterdeck's DESQview, multitasking was becoming increasingly popular. (For a time, AST bundled DESQview with its Rampage boards as an inducement to get users to buy more Rampage boards, which they were sure to do if they used DESQview.) DESQview's multitasking required expanded memory but only worked with AST's EEMS expanded memory, not the LIM EMS 3.2 expanded memory. Therefore, responding to user pressure, the EMS specification was redrafted to incorporate most of AST's EEMS features.

The new LIM EMS 4.0 specification that emerged ignored the fact that, except for the exclusion of a few lesser features, it was an old copy of AST's EEMS. Copying EEMS ruffled more than a few feathers because Lotus, Intel, and Microsoft had demonstrated little genuine interest in the old 3.2 specification. However, the world had a new specification, and it's incredible what the right names on a piece of paper can do for the acceptance of a standard.

Under the new EMS/old EEMS specification, access to expanded memory was not limited to just the page frame but could include all the memory available to the CPU, starting from 0K right on up, or any part thereof. Fully implemented, the CPU owns nothing; the CPU has to look to the memory manager for everything (Fig. 3-4). All memory available to the system, either for filling conventional DOS memory addresses (up to 640K) or for use as expanded memory accessed though the page frame, would come from a common memory pool (Fig. 3-6). In many ways, this method affords a perfect solution, but alas we live in an imperfect world. Implementation has remained spotty.

At the one extreme, AST built and marketed a 286 machine with no user RAM on the system board at all (0K). All of the system's memory is installed on one or more plug-in memory boards. Without at least a partially populated memory board, you couldn't even boot the system because you wouldn't even have a place to load the operating system. No responsible vendor wants to sell you a machine that won't even run, so not surprisingly, AST sells this system only with at least one memory card (minimum 512K) installed.



3-6 RAM for all uses—Conventional and Expanded Memory—is supplied from a common pool. The increased flexibility afforded by this scheme—but not provided by many computers even today—is critical, especially to multitasking and many 4.0 EMS applications.

As a purely practical matter, you cannot actually swap out your entire conventional memory area. If you tried swapping out even the small part of the kernel DOS 5.0 leaves behind when it relocates the bulk of itself to the High Memory Area, the system would die almost immediately. There are some other mundane system functions that cannot be swapped out either: addresses for things like parallel and serial ports, disk drives, and other devices. Still, with DOS 5.0, the total of these requirements is under 32K, so ideally you would like to be able to swap out almost all of the memory below 640K.

Looking at the other end of the spectrum, however, there still are some real dinosaurs on the shelves. The users of these systems never will be able to fully tap the potential of this new technology at all. Some users might be able to use the technology but only by wasting up to a full megabyte of RAM. IBM's PS/2 model 50s and 60s fall into this category. The entire megabyte that comes standard with the system is wasted if you want to avail yourself of the full LIM 4.0 EMS capability by backfilling down from 640K. First, the entire contents of conventional memory must be copied into expanded memory.

Then, using a software switch (in the PS/2 Model 50 or 60, memory is controlled by software rather than by DIP switches), the system board memory is disabled, or turned off. You must turn it off. The system doesn't allow you to free the system board memory for assignment to the expanded memory pool. You also can't have two blocks of physical memory answering at the same addresses.

After the system memory is disabled, the expanded memory containing the copied contents then is swapped back to take its place. From that point on, you have full LIM 4.0 EMS functionality, including backfilling clear down to 0h. However, the memory is installed on the system board just goes to waste. Either the CPU owns it or nobody does.

There are all sorts of middle-of-the-roaders offering varying degrees of flexibility. Typically, these systems don't start at 0h but somewhere up around 256K, allowing you to swap conventional memory for expanded memory from 256K on up to 640K. Enough systems have taken this tack to lead many users to the erroneous conclusion (fostered by much of the popular computer press) that the potential for backfilling, or swapping-out, of conventional memory for expanded memory is limited to that 384K block, which simply is not so. This amount really is an arbitrary number, a convention based only on statistics, nothing more.

Perhaps it should be thought of in terms of starting from 640K and working down, instead of working up from any starting point, because only the top is fairly (though not completely) fixed. Memory above 128K could be disabled on the original Compaq Deskpro. Even the old 8088 PC allowed memory above 64K to be disabled. The Epson Equity Plus allows any and all memory installed on the system board to be switch-disabled down to, and including 0h. Various members of the IBM PS/2 family allow disabling of system board memory via software switching.

Note, however, the term *disabled*. As pointed out before, disabled usually means just that: locked out, wasted in the final scheme of things. You don't necessarily have to install the full 640K on your system board—assuming you have socketing for it—just so you can switch it out and waste it. Given any choice in the matter, you usually can get by with very little memory (or in some cases, none at all) installed there, leaving it to your extended/expanded memory hardware and software to fill in all the gaps with manageable blocks.

And in the center ring . . .

The situation was like trying to watch a 3-ring circus. While the 286 and expanded memory debate was going on, the 80386 came on the scene. Functionally, the 386 looked little different from the 286 AT at first glance. Not surprisingly, the initial reaction to the 386 machines, in many circles, was an overwhelming "Ho hum." Indeed, despite the chip's full 32-bit processing power, early 386s offered only the standard 16-bit AT (ISA) bus and little outward evidence of improvement (other than faster clock speeds). Most 386 machines also included at least one 32-bit slot; however, there was no industry standard 32-bit bus architecture, so these slots were, for the most part, of use only with special proprietary memory boards.

Intel, however, had done it right this time. Among other things, the near-fatal defect that had plagued the 286 had been completely overcome. The promise of extended memory and a protected mode had been fulfilled.

Still, the full impact would have to await the development of a whole new genre or applications software to be appreciated. In the meantime, however, the 386 had other capabilities that had not been seen before. One of these capabilities allowed managed memory to be mapped to any unused address space, including spaces that had never been used before such as isolated pockets of genuine DOS-addressable addresses between 640K and 1024K.

Seizing on this opportunity, two software developers in particular—Quarterdeck (the DESQview people) and Qualitas (386MAX)—quickly developed powerful new 386-specific memory management packages to cater to this special ability. The new memory managers provided expanded, extended, and conventional memory from a common memory pool. They were able to map memory to these previously unused blocks of address space. Finally, they provided the mechanism for relocating TSRs and device drivers to these areas, thus freeing the space they had taken in conventional memory for bigger applications and/or data files.

By today's standards, those early 386 memory managers were rather crude; however, up to nearly 200K of new DOS memory was immediately available, as well as the promise of even bigger and better things to come. The 80386 could use extended memory. It could take the extended memory—up to 4 gigabytes of RAM—right in stride, without the paroxysms suffered by the 286. In terms of hardware, the pieces were all finally in place.

4

CHAPTER

Expanded memory: something for everyone

While much of the original luster has dulled, most of the emphasis today moved to the arena of extended memory. Expanded memory was the first glimpse any of us had of life beyond 640K. It remains—and will continue to remain—the only form of added memory available to all DOS users, regardless of the hardware platform they might be working from, whether one of the original IBM PCs or the latest *i*486.

Today, the future clearly lies in extended rather than expanded memory. Expanded memory, however, will continue to play an important role—as long as there is DOS—for three reasons:

- The ability to have expanded memory on any machine opens up a far larger potential market to software developers than is available for any hardware-specific products.
- Under the LIM 4.0 EMS specification, memory from the same memory pool can be used to backfill conventional memory address space.
- LIM 4.0 EMS memory also can, with proper management, be mapped to upper memory blocks on 386 and higher machines. Upper memory then can be used to load device drivers and TSRs, freeing lower memory for use by applications.

These three functions, while all founded on the use of EMS memory, are so different conceptually and so different in their implications that I will deal with them in separate chapters. This chapter will look at expanded memory as it was originally intended and is still used commonly. To appreciate this facet of the jewel, you need to look back to the origins of expanded memory as implemented in the world of DOS.

In the beginning

Lotus was one of the prime movers behind the introduction of expanded memory to the DOS arena. It was responding to pressure from users unable to live and work with larger Lotus spreadsheets within the 640K memory model DOS allowed. The problem was not a lack of space to run their program code. The problem that users suffered from was insufficient space for storing data.

This problem is critical because it is the sticking point. The issue was space to store data temporarily when it was not actually being used. The thought—even the hope—of running programs in the background or implementing any of the other functions associated with EMS memory today never entered into it. EMS was a quick fix for an immediate problem, nothing more.

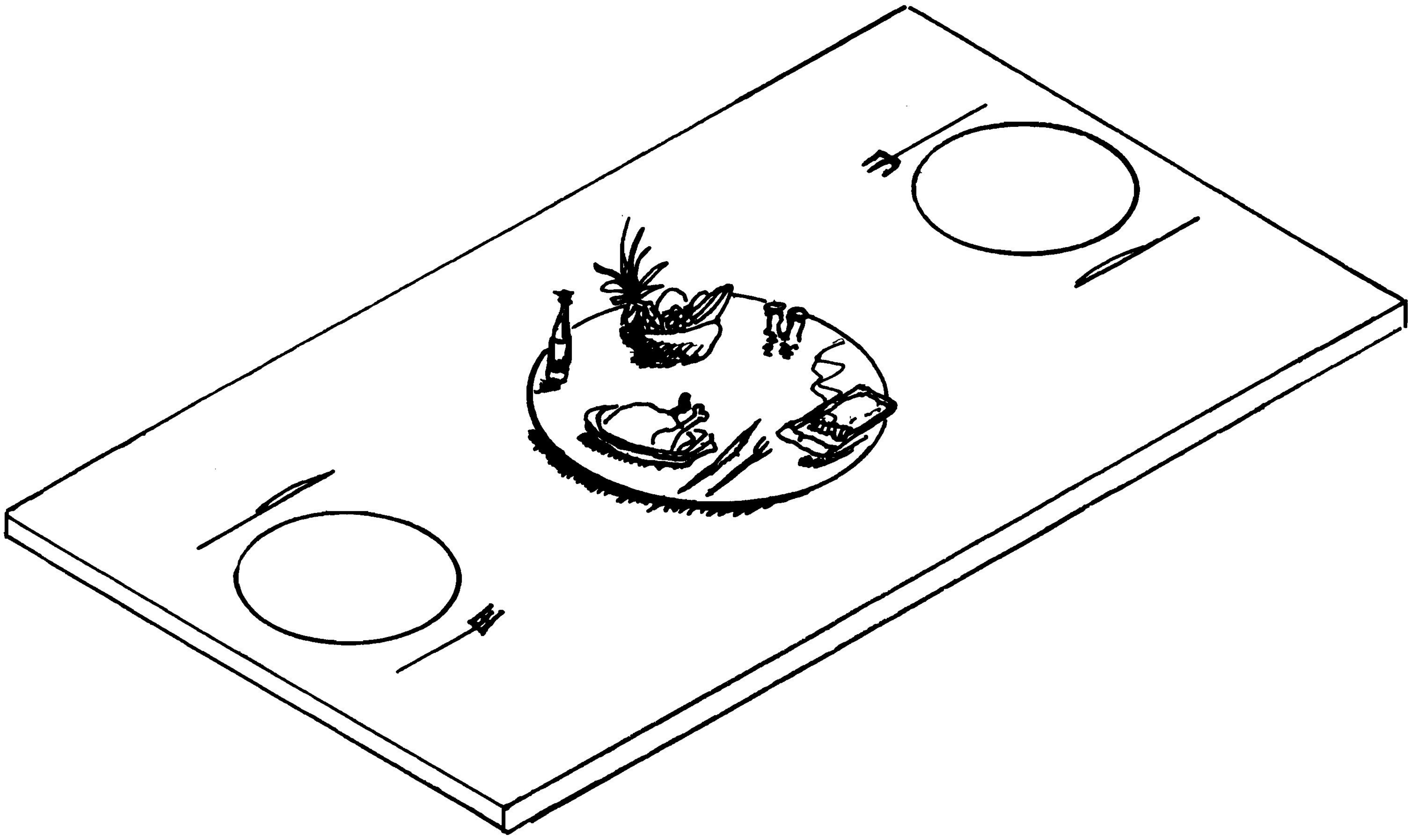
The original expanded memory scheme, borrowing bank switching technology from the mainframes, was supposed to allow users to access up to 8 Mb. In practice, that amount proved optimistic, but it was enough to get things rolling at least. Lotus, Intel, and Microsoft sat down and wrote a formal specification to define this expanded memory scheme.

Actually, Microsoft, rather typically, did not even join in at first. This fledgling idea received a tremendous boost when Microsoft did put its name to it. IBM most surely was consulted in all of this development and, just as surely, had its say in it. After all, the blocks of addresses needed were in an area IBM had big RESERVED signs planted in. Whatever role it might have played, IBM did not put its name on the specification. IBM did not even officially recognize expanded memory before DOS 4.0 came into being.

Back in Fig. 3-1, I depicted expanded memory as it was originally configured. Extended memory passes small portions of memory through a 64K page frame on a revolving access basis which is managed by an expanded memory manager. Up to a total of 8 Mb is accessible.

A little dinner music, please

In operation, expanded memory has probably been best compared to something like a lazy Susan on a dinner table. (While the analogy is flawed in that no physical motion is involved, I think it's close enough to serve our purpose here.) So imagine yourself at a table with a huge lazy Susan in the middle (Fig. 4-1). The



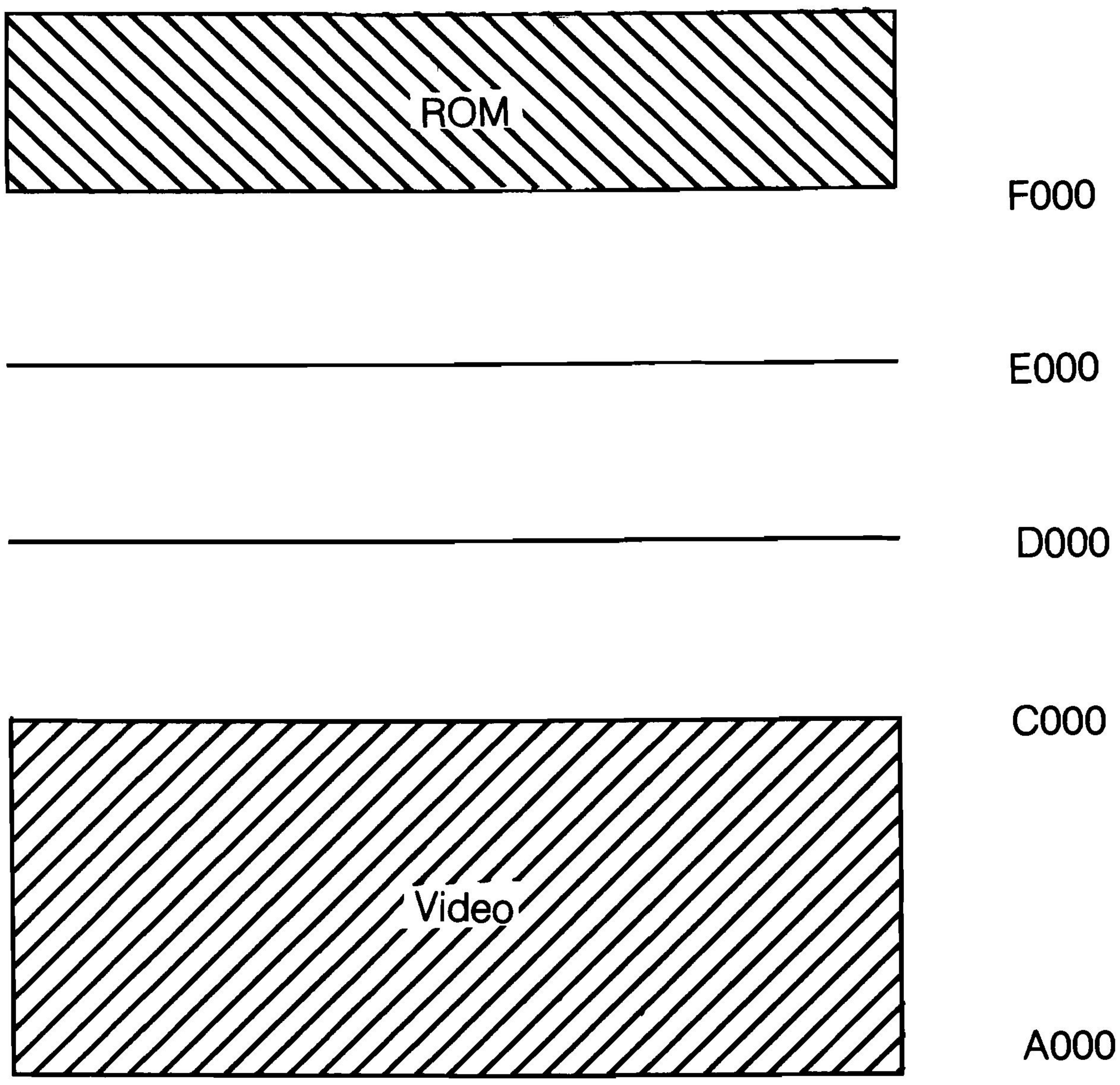
4-1 Expanded Memory sometimes is compared to a lazy Susan. When virtual memory is used, different data is located at different places on your hard disk.

olives and cranberry sauce are in front of you; the turkey and dressing is out of reach at the far side. You spin the lazy Susan and spear a piece of turkey with your fork. The olives and cranberry sauce still are there on the lazy Susan even though at that point they are out of reach. When you want them, all you've got to do is spin the turntable again until they come back into reach.

Adding more expanded memory is like building a bigger lazy Susan. The bigger it is, the more goodies you can pile on and bring spinnning within arm's reach. With a little practice, even a blind man could find what he was looking for, as long as everything was always in the same place on the merry-go-round. Given the obvious loading options expanded memory opens up, where things were stuffed during yesterday's work session or even where they are this minute really doesn't matter as long as memory manager knows where they are. The memory manager can bring them back to someplace where DOS can see them—a window or a frame of some sort.

This frame—sort of like having to reach through a croquet hoop to get at something on the lazy Susan—is called a *page frame*. In the case of expanded memory, the frame is simply a block of DOS accessible addresses above 604K but that by definition, must be below 1024K to be accessed through DOS. (The High Memory Area between 1024K and 1088K, discussed later in this chapter, is a special case and does not violate this rule.)

DOS, historically and for reasons that are no longer really valid or important here, is generally broken into 64K segments: 10 for the user (0h through 9h) and 6 for the system (Ah through Fh). Not surprisingly the block generally allocated as the page frames (the window to expanded memory) is a 64K block. As you can see from Fig. 4-2, there were blocks beginning at C000h D000h and E000h under the 3.2 specifications. (Not all boards supporting the 3.2 EMS specification supported the entire allowable range.)

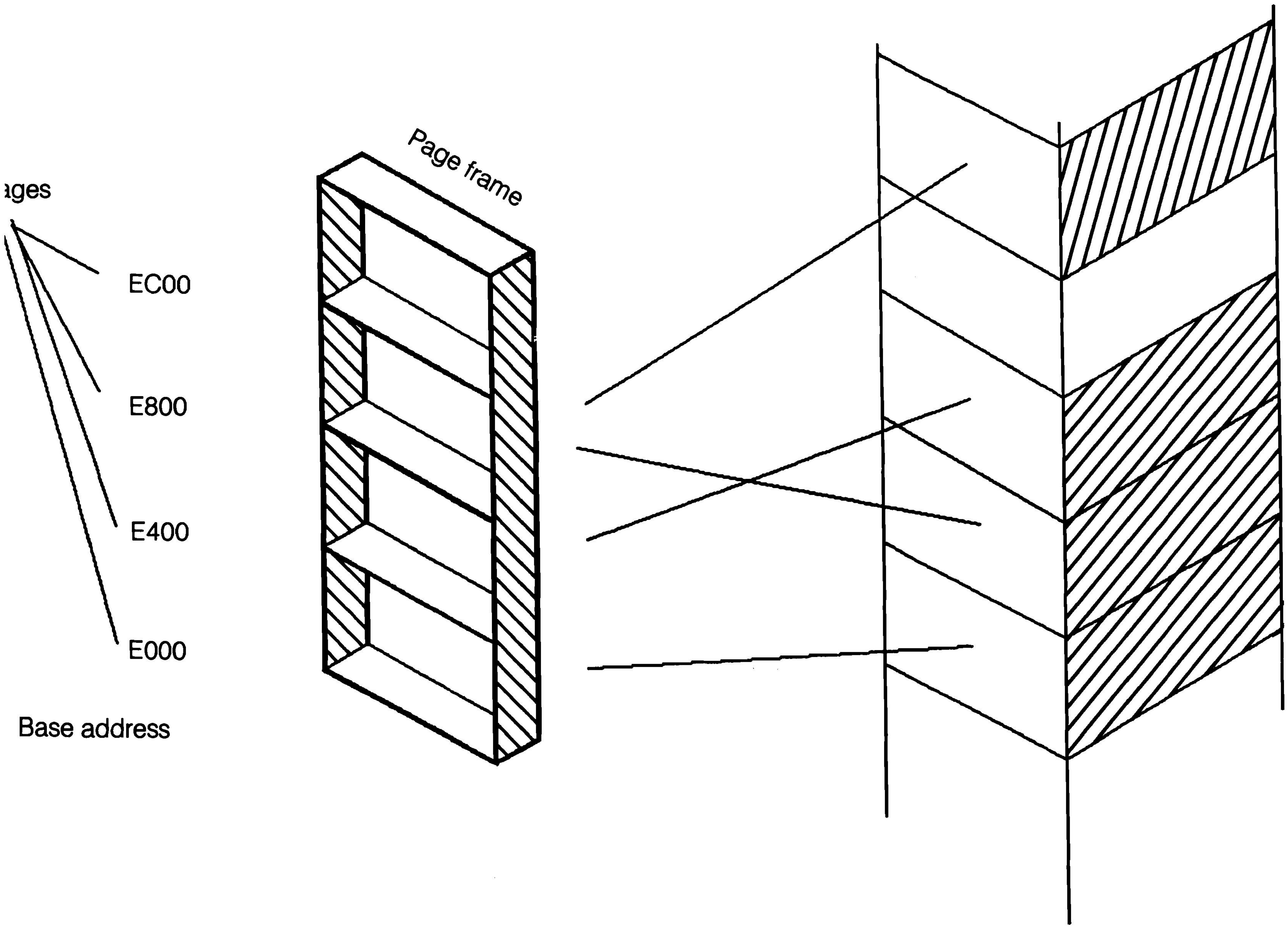


4-2 The page frame for expanded memory generally is installed at one of three addresses: C000, D000, or E000. However, it need not use these exact base addresses as long as the base address is a multiple of 16K and there is 64K of contiguous memory available beginning at that address.

Referring to Fig. 4-3, note that the 64K block, or frame, is broken down still farther into something called *logical pages*. Each page contains 16K with four pages to the frame. The page frame base addresses do not have to be multiples of 64K, but can fall at multiples of 16K, using any 16K block of addresses above 640K not occupied by ROM or RAM. The actual physical block of memory the CPU sees at those addresses can be anywhere in the 32 Mb of actual installed and addressable RAM chips that expanded memory allows (under 4.0 EMS).

No matter where the memory actually is located, DOS sees only calls to legitimate DOS addresses when you try to access the memory. Some form of memory manager must, in effect, spin the lazy Susan to bring the actual block of memory that's called for back into the window. Having done its job, the EMM drops out of the picture until the next time it has to spin the lazy Susan.

Remember, no actual physical motion has taken place. It is an illusion, in much the same way as call forwarding can intercept telephone calls incoming to your home or office and invisibly reroute them to you someplace else. Where you



The page frame is divided into four 16K logical pages, as shown here where the base address for the frame is E000.

are doesn't matter. All that matters is the information is exchanged and everybody (hopefully) is happy.

To Lotus users, memory management meant they could have huge spreadsheets—spreadsheets requiring several megabytes of data space. With larger spreadsheets, you're usually only working with small portions at any given time anyway. For such applications, a page frame that allows for ready access to any selected part or parts of the data on a rotating basis is often all that's really needed.

Swapping chunks of a large spreadsheet through a small window was not necessarily the issue, just the ability to do it. Expanded memory could do it. As originally defined, expanded memory was intended solely to increase the memory available for an application's data space. The original 3.0 specification left much to be desired and was soon modified. There was a short-lived 3.1 specification that included some additional features that Lotus needed. A 3.2 EMS specification emerged then, taking some of the rough edges off of the earlier specification but adding little more to it.

Prior to its DOS-Extended 286 release 3.0 in 1989, Lotus had never updated its product to support EMS features beyond those incorporated in the 3.1

expanded memory specification. Lotus' inability to access more than a quarter to possibly half of the 8 Mb allowed by the early EMS specifications was sometimes cited by some as reason for the larger 32 Mb limits of the 4.0 EMS. Lotus, however, never changed the internal memory management functions to take advantage of the greater memory allowed even under 3.2 EMS.

Using expanded memory

When and how you use—or can use—EMS expanded memory depends on several factors. The use of EMS memory to backfill conventional address space, as discussed in chapter 5, is completely invisible to software. When we move out of the conventional memory area to dip into the memory pool available through the page frame, however, it is a different world. Just because you add EMS memory to your machine does not mean any of your favorite programs you're using are going to use it. These days the odds are fairly good. Even then, in many cases, you must have one of the later releases of the program.

The use of any managed EMS memory other than to backfill conventional memory addresses requires that programmers make specific provisions for such usage in their programs. Increasingly, software developers have provided for such usage, driven by competition and the demands of a more enlightened user base as our insatiable appetites for memory, more memory—any kind of memory—continue to grow.

In some cases, increasing available memory is an easy fix; in others, it does not seem so easy, but it's been happening. At the time I wrote the first edition of this book, I only had a small handful of TSRs that would run happily above 640K. Today, the numbers are reversed. With upper memory becoming overcrowded these days and with the ready availability of expanded memory, an increasing number of programs have found ways of reaching through the page frame.

To access memory through the page frame the program has to:

1. Determine if expanded memory actually exists on the system.
2. Determine if there is enough expanded memory to meet the program's needs (Function 3).
3. Allocate expanded memory pages as needed (Function 4 or 18).
4. Get the available page frame addresses (Functions 2 and 25).
5. Map in those expanded memory pages (Function 5 or 17).

Only after it has done all these steps can a program read, write, or execute data in expanded memory as though it was conventional memory, which is where the program loads and runs if it is unable to complete the first five steps. There is, however, one more step. The program also has to have a mechanism for returning whatever expanded memory pages it used to the expanded memory pool before exiting (Function 6 or 18). The program has to clean house, so to speak.

I included function numbers here only to illustrate the point that these are indeed formal functions covered by a formal set of rules as spelled out by the specification. For a complete listing of the functions, see Appendix A.

The most important thing to see from these steps is that the rules pose no restrictions on the actual program other than the normal rules of DOS. Where the program is allowed to load and run and how it exits are, in most cases, all that needs to change to utilize expanded memory.

The above list, however, includes only the bare minimum of functions a programmer must include to utilize expanded memory. It is by no means a complete list of the tools available to enhance the capabilities of software running in expanded memory. Anyone interested in investigating these functions in greater detail should consult the actual specification document.

Expanded memory that is—but isn't—what it seems

Through the years, humanity's inventive nature has rarely let a little thing like obstacles stand in its way for long. Even as the PC began to slip its bonds and leave its toy image behind, programmers had found ways—many borrowed from the mainframe world—to make whatever actual resources they had available look and act like more.

This stretching of resources has been especially true in the case of memory, which is a finite resource with limits beyond which your hardware cannot go. As the size of programs grew, programmers were forced to look for solutions that would allow their programs to run even though the programs were larger than the amount of space (RAM) available on a typical user system.

In the inventory of ordinary resources, there generally are two principle kinds of user memory. There is RAM, and volatile medium but the medium of choice for temporary storage. There also is some form of nonvolatile mass storage, generally a hard disk with a lot of megabytes. Just as, with a little clever programming, RAM can be made to look like a storage disk (RAM disk) part of a storage disk can be made to look just like more RAM—you, in many cases, really wish you had but often simply can't afford.

The ability to simulate RAM on a storage disk has led to the development of a group of expanded memory emulators. One such emulator is incorporated in Turbo EMS, which, although primarily a memory management tool, bills itself as "the affordable alternative to expanded memory boards." Do not confuse these emulators with EMS *emulation* in the sense the term is used in conjunction with DOS 5.0's new EMM386.EXE EMS emulator, which borrows extended memory and presents it to software as 4.0 EMS memory.

Turbo EMS, or any of a genre of comparable products, deals in virtual memory; it simply swaps the contents of RAM to disk when you don't have enough real memory to go around. Even though several such emulators boast 4.0 EMS support, the mere fact of swapping to disk precludes any real multitasking, memory mapping to upper memory blocks, etc. In terms of performance, you might as well be back under 3.2 at that point (which is why I am including it in this chapter rather than discussing it as a legitimate 4.0 EMS issue).

I don't mean to pick unfairly on Turbo EMS or the several other comparable products, but rather only to put them in their proper perspective. Even such better known packages as Software Carousel and even DESQview will swap to disk when the well runs dry. Swapping to disk is the only way the new DOS 5.0 task swapper knows. Because there are a number of programs that can or do use what is euphemistically called virtual memory, I think the point needs to be made while on the subject of expanded memory: virtual memory, by definition, is not for real.

I am not saying the use of virtual memory does not have a legitimate role in many, if not most, user environments. On the average, I probably use it once or twice a week myself when I have to open one more multitasking window than the RAM I have supports. In those cases, even at the price of RAM today, the few seconds 2 Mb of additional RAM save me doesn't justify the cost. When and if it is justifiable, the only affordable alternative will be the real thing.

Only a stopgap

At most, expanded memory, as it first entered the DOS scene, was a stopgap measure by an infant industry trying at that point to define its legitimate place. There are problems today that still are rooted in the history of expanded memory and a failure of the EMS specifications—even to the present—to define a minimum hardware level that can lay claim to support these specifications. Expanded memory, however, was a turning point and the beginning of a movement no one had foreseen.

5

CHAPTER

EMS memory: the dawning of another age

From the beginning there were many people who said the LIM 3.2 EMS specification had not gone far enough and essentially was written only to get Lotus off the hook. Such things as multitasking, as many of us know it today, simply could not have happened.

At the forefront of those who said the LIM specification had not gone far enough were a number of other vendors of both hardware and software. Although EMS was strictly a software standard, hardware was required to implement it, which meant you first had to have the hardware. Among hardware vendors, expanded memory triggered a whole new gold rush and a variety of new products.

Intel was in there at an early stage with its AboveBoard, which quickly spawned a generation of AboveBoards and look-alikes, or clones. Because Intel had been instrumental in developing the LIM EMS specifications, its boards, not surprisingly, provided rather scrupulous hardware support based on that standard.

AST Research was in the game rather quickly too; however, it went a rather different direction with its Rampage boards. While staying within the rules set down by the 3.2 specification and remaining fully compatible with it, AST went one step—a giant step—farther. AST developed its own superset called EEMS (Enhanced Expanded Memory Specification), which added a number of novel and, at the time, radical features.

Most radical of all was the notion that all of the memory should be software switchable en masse under the control of the memory manager—not just in the 64K page frame, but also through the entire range of address space within its grasp. At the time, 256K typically was the maximum capacity of a fully-populated motherboard. To add any memory beyond that required an expansion board.

AST's EEMS boards were expansion boards. They could take you to 640K and, if you had enough chips, some expanded memory as well. Because the boards were AST's radial EEMS specification, any memory you had beyond 246K limit of the motherboard ultimately was under the control of the memory manager and could be switched in and out at will.

Interestingly, no one—not even AST—initially knew what real use some of the advanced features they were touting might be put to. Before long, a then obscure group, calling themselves Quarterdeck Office Systems, came along and showed the world what EEMS was good for: multitasking. And it was an idea whose time had come.

Quarterdeck had achieved real multitasking, not just context switching the way it could be done with Microsoft's Windows or Software Carousel. This multitasking could be done even with an old 8088 PC. DESQview, teamed with the EEMS boards, could swap whole running programs—as big as 384K or more—in and out of conventional memory and keep them running in the background while you had some other process running in the foreground.

The critical difference is that programs shuttled to the background could keep right on running, doing real work rather than just sitting in limbo, as with context switchers. You, however, had to have an EEMS board to make DESQview work with any combinations of programs that totalled more than 640K (including the operating system and DESQview's own overhead).

In the meantime, many Lotus users were again unhappy because, as a practical matter, they could only access about half of the promised 8 Mb. Unfortunately, neither the boards Intel was selling nor the early EMS specifications that bore its name supported real multitasking. They supported context switching but not multitasking.

To make matters worse, by then Intel had a new chip, the 80386, that was perfectly suited for multitasking—multitasking as only EEMS supported it. Something clearly had to be done about it, so Lotus, Intel, and Microsoft finally produced a new document. Although seemingly based entirely on AST's EEMS specification and incorporating most of the features available previously on AST's EEMS boards, the document was called the LIM 4.0 EMS specifications. AST's EEMS functions in some ways still exceeded those incorporated into the LIM 4.0 EMS specification.

LIM 4.0, the operative specification today, provided new and better utilization of expanded memory for such things as:

- More effective use of expanded memory for spreadsheets users
- The ability to load and run memory-resident programs (pop-ups or TSRs) in expanded memory
- Families of applications using shared data in expanded memory
- Larger RAM disks, print spoolers, and disk caches
- A 32 Mb limit, instead of the old 8 Mb
- Multiple programs in expanded memory, simultaneously and with better performance
- Allowing more than 64K to be addressed at one time
- Allowing memory to be addressed both above and below 640K
- Allowing for alternative mapping registers for high-speed bank switching

The 4.0 specification maintains backward compatibility in the sense that programs written to conform to and fully implement the new standard will run on hardware supporting the earlier 3.2 EMS or EEMS variants. In many cases, new device drivers might be required. In most cases—with the probable exception of hardware supporting the EEMS standard—performance, however, will not match that attainable with hardware designed to support the later 4.0 specification.

To software developers, the 4.0 specifications offered 15 new functions and 39 new subfunctions. The benefits from these new functions are not all immediately obvious to the user; however, in the overall scheme of things, they are, the heart and soul of the 4.0 specification. The new functions include:

- Multiple page mapping—a real plus for better performance in general and data protection
- Dynamic growth or shrinkage of the amount of expanded memory allocated, allowing memory that is no longer needed to be returned to the pool for availability to other applications, or to increase the amount used after the initial allocation
- The naming of data "handles" where data is to be shared between two or more applications loaded and running simultaneously
- Far jump and far call simulation—the ability to run code in expanded memory
- The ability to copy or exchange (swap) a region of memory from conventional to expanded, expanded to conventional, or from expanded to expanded
- The ability to map more than 64K at a time and/or to map into conventional memory (This function is probably the most important feature of 4.0.)

This last function is where the backfilling capability comes into play, with the ability to swap a whole chunk (typically, but not limited to, the area above 256K)—running code and all—and, where multitasking is available, to keep it running in the background.

Under the new specification, expanded memory took on a whole new meaning. The 1 Mb computer suddenly had become more host than master in a world expanded to embrace as much as 32 Mb.

The 4.0 backfill: mapping conventional memory

The promise of a quantum leap in total memory was not what fired the world's imagination. Few users even now have, need, or even want that much memory. The ability to backfill conventional addresses from 640K down to some point (typically around 256K) with managed memory drawn from a pool of memory was what had everyone excited.

Most of the pre-LIM 3.2 boards allowed you to assign whatever amount of memory needed to bring your system to the full 640K of user space. When 3.2 came into being, the better boards allowed you to assign any memory you had left over as expanded memory.

Under 3.2, when you assigned a block from an expanded memory board to bring your system up to full capacity, the block was assigned to the CPU alone. Using DIP switches, software, or some combination, you gave that memory to the CPU. From that moment on, the CPU owned it in the same way it would have owned the extra memory, an ordinary expansion slot multifunction card, or a normal memory board. When 4.0 is fully implemented, however, that backfilled memory is only loaned to the system on a revolving basis (like the lazy Susan analogy in the last chapter). The memory, still is owned by the expanded memory manager, not the CPU. Because it is only on loan, the memory can be taken back and swapped—the whole 384K or whatever in one fell swoop—in addition to whatever you might do with any or all of the 16K logical pages of expanded memory accessed through the addressed C000h through EFFFh.

The 256K floor actually is an arbitrary number based more on the fact that few computers—then especially—allowed for memory management below that point. In many cases, at least the first 256K on the motherboard was essentially fixed. It was someplace to load DOS and possibly some other essentials and whatever else just happened to get loaded there.

There is little reason today not to allow backfilling everything except possibly the first 64K (000h to 0FFFFh)—especially with DOS 5.0's ability to load most of itself in the High Memory Area (HMA) above 1 Mb. This ability, however, depends on the design of the underlying hardware.

Originally, the backfilling feature was available only when an EEMS expanded memory board supplied memory below 640K. Today, it is available by design on most better-designed computers. Don't make the mistake, however, of just assuming—or taking some salesman's word—that this capability is built in.

With boards fully supporting the 4.0 EMS, backfilling is entirely different from any expansion memory below 640K that is supplied by boards that really only support the 3.2 specification. Under 3.2 EMS, no bank switching was allowed below 640K. Although expanded memory boards could fill any gap to present a contiguous 640K block, that memory became dedicated memory locked into specific addresses and owned by the CPU.

Assuming your computer and/or expansion boards fully support the 4.0 EMS specification and you have a 4.0 driver installed, no program can tell the difference between backfilled memory and hardwired memory. As long as you are able to load your programs within a range of continguous conventional memory addresses, what you do with them after that doesn't matter. With 4.0 EMS, you can switch fresh blocks of memory of whatever size you need in and out of those same conventional memory addresses, until you run out of memory.

But not for everyone

As much as the potential of backfilling opens up exciting new horizons, you can't just run out, buy a 4.0 EMS device driver for your old 3.2-era hardware, and get 4.0 capability across the boards. You can't even do that with some newer hardware that claims to support the 4.0 EMS specification. Unlike DOS upgrades that many times do add new capabilities to old machines, there is often little to be gained by simply upgrading the expanded memory manager. You, however, should not automatically rule out that option, especially with 386 machines where other issues are involved.

No software can increase the physical capabilities of a piece of hardware. Software can be rewritten, written over, and improved, but hardware is another matter. Sometimes, however, better software can unlock capabilities that were in the hardware all the time but just not utilized. DOS has often done that; however, it has only been able to make an old machine work better where and when the operating system was able to access and implement unused hardware capabilities. Unfortunately, boards designed to support the original 3.2 specification generally do not have the hardware capability to support backfilling.

With few exceptions, earlier expansion boards that supported the 3.2 EMS Specification did allow up to 384K of whatever memory was installed to be used to backfill any memory deficiency below 640K. That memory, however, became dedicated memory. Those boards, typified by Intel's earlier AboveBoards, did not support the swappable backfilling capability defined in the LIM 4.0 EMS specification.

This inability to perform backfilling does not preclude bringing the benefits of backfilling even to an old 8088 PC. Whatever add-in hardware you install must support backfilling and be able to remap any motherboard memory it finds between some hardwired point and 640K. The hardware also has to be able to take control so it can effectively swap the whole block out and swap another in that block's place. The LIM 4.0 EMS specification, which is what all of this discussion revolves around, is strictly a software specification and has nothing to do with hardware.

The sheep from the goats

Admittedly, you cannot have expanded memory without both hardware and software. There, however, is no EMS hardware specification in existence. At one time, such a specification did exist but Intel decided to let manufacturers design hardware as the manufacturers pleased so long as the software driver provided a standard interface. The decision is up to each individual manufacturer whether to support or not support the software standard. The result of this lack of standardization is that there are major differences in the degree of which various vendors support the most important features of the LIM 4.0 EMS specification.

As long as they don't do anything that is incompatible with the 4.0 specification, vendors can and do claim to support 4.0. In many cases, the hardware they are selling is still of 3.2-era design, supporting none of the added features covered by the later specification. Those hardware shortcomings cannot be overcome by software.

No matter what you do or what the salesman might have told you, a 3.2 board is still a 3.2 board. Most add-in memory boards can be used to deliver either expanded memory, extended memory, or a mix of both. The issues when buying a board you intend to use for expanded memory are different from those for extended memory.

Registers: real and fake—and often missing

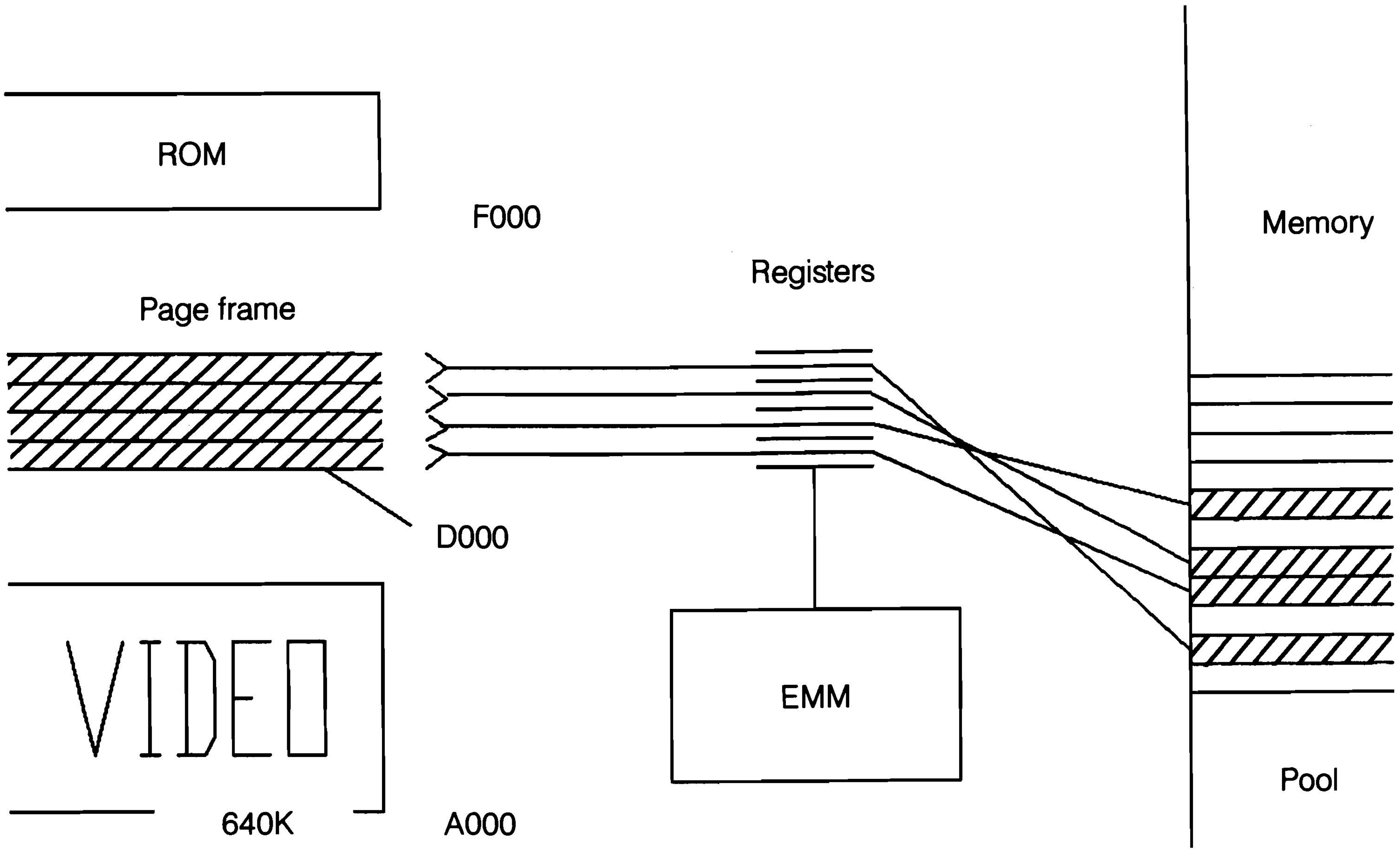
One important factor in the delivery of EMS memory—particularly full 4.0 EMS memory—is something called *registers*. Registers are where the data that keeps track of all the bits of swapped-out code is actually stored.

Properly called *mapping registers*, the function of each register is to point to a 16K block somewhere in a memory pool that, under the 4.0 specification, can include up to 32 Mb of expanded memory. This will be mapped in and out of logical pages that now, at least in theory, have addresses anywhere within the DOS's 1Mb province.

Mapping registers are like call forwarding for your telephone; nobody's home at your number but you can be reached someplace else. This feature is the whole key to expanded memory: the interception of address requests and their redirection to parts unknown to DOS and otherwise inaccessible to it.

In operation, the expanded memory manager simply dips into the pool of memory available to it and finds available blocks of memory as needed for each task. Having done that, some sort of record must be kept, so the manager can find those blocks again whenever they are needed. The registers provide that record to remember and be able to point the way back. Each register corresponds to the base of some 16K block of addresses—not memory. Calls sent to those addresses are directed to some corresponding block known only to it somewhere in expanded memory.

Under the earlier 3.2 EMS specification, there was really only one usable 64K page frame, encompassing only four logical pages of 16K each (Fig. 5-1). There was only one pointer, or register, per logical page. If you had four registers, you had it made.

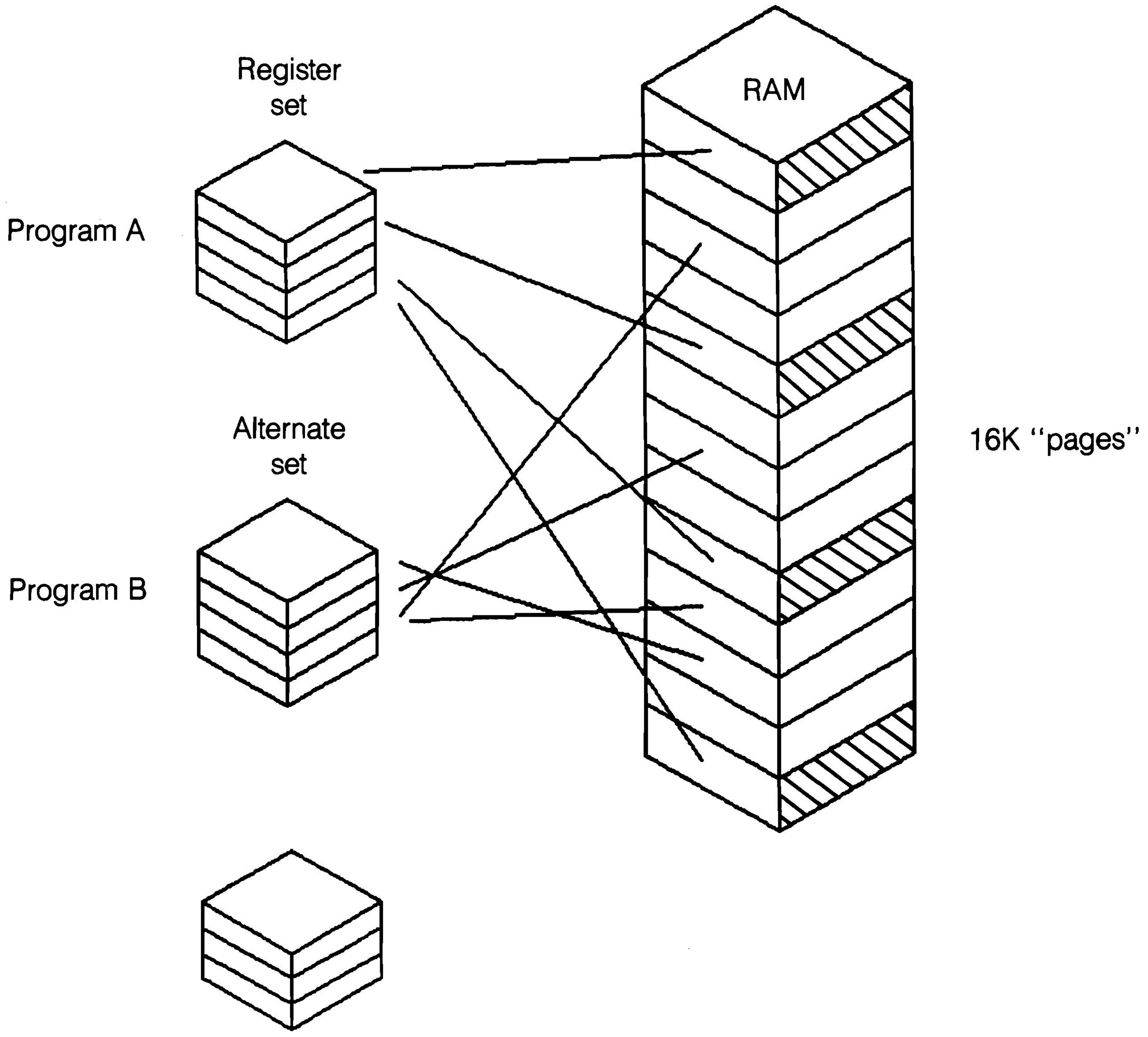


5-1 Putting the pieces together, you see the relationship between the page frame (divided into four 16K pages), a set of registers controlled by the Expanded Memory Manager, and the 16K blocks (pages) of memory that the individual registers point to in the pool. Note particularly that the actual blocks addressed are not shown to be contiguous. Typically, they would not be, although they conceivably could be. For simplicity, only four registers—all that was needed under the old 3.2 specification—have been shown here. Most boards that properly support the 4.0 specification have increased the number of registers per set to 64 to allow mapping to conventional memory address spaces.

The whole group of registers—four in original 3.2 EMS context—make up something called a *register set*. One register set can point to several 16K blocks somewhere in expanded memory. If you want to access expanded memory for a different program, or even a different block of data for the same program, you have two choices:

- Hunt up a different set of data that the registers must contain to access whatever other blocks the EMM needs and write that data over the data currently in the registers.
- Have alternate sets of registers and simply switch sets.

In that scenario, the obvious choice—if you have one—is to have alternate sets to be used as switching sets. This method is much faster than having to hunt for and rewrite data every time for you to make a switch. In Fig. 5-2, you can see how alternate sets come into play. Note that the registers are shown even crossing over each other. Although some set of rules buried in the software determines how these blocks are selected, the selection might appear to us to be totally random. It doesn't matter though as long as a register set retains a record of what is where and belongs to which program.



5-2 The use of alternate register sets is illustrated here. The register set for Program A is pointing to four 16K pages, but an alternate set already is loaded and ready to switch instantly to a different set of pages for Program B, while a third register set is available, if needed. Again for simplicity, sets of only four registers each have been shown here; however, in actual practice, under 4.0 EMS, 64 registers per set are required for full implementation.

If you're wondering why I'm still talking about obsolete technology now in the face of the LIM 4.0 EMS specification, it is to make a point. The new specification provides many new memory management functions not available under the earlier specification. A fair percentage of the expanded memory boards marketed as supporting the LIM 4.0 EMS specification, however, do not support many—if any—more features than were included in the earlier specification. The same is true of many of the EMS device drivers still being distributed.

Supporting the 4.0 EMS specification *does not* necessarily mean supporting all the features and functions offered under that specification. All it means is that the vendor isn't doing something weird that doesn't conform. The wary buyer needs to know and understand the difference. Even conformity isn't necessary considering some of the weird things IBM did when it first supported the 4.0 EMS in a way that was totally incompatible with everyone else's interpretation.

As long as expanded memory has existed, registers and register sets have been there. The significance of registers and register sets, however, becomes a matter of special concern with the 4.0 EMS (or beginning with AST's EEMS if you were really on top of things) as we move up to multitasking. Under 3.2, you could not do multitasking; with EEMS or 4.0 EMS you can. Registers are the key. It's that simple.

The difference comes down to how many registers you've got—how many make a set. Four was all you ever needed under 3.2 (four 16K logical pages totally on 64K page frame).

AST was the first company to increase the number of registers. It needed the additional registers to support its then radical EEMS scheme, which took the basic EMS concept and extended it, opening the door to memory at any address within DOS's entire megabyte of address space. In 16K chunks, that takes 64 pointers or registers (1024 ÷ 16 = 64). Realistically, there are some practical limits here, at least as we are able to juggle our memory usage now.

Limitations or not, however, the critical issue now is the ability to swap conventional memory in and out, not just four pages in a page frame but a lot of additional pages from 640K on down. If you only swap from 640K down to 256K that requires an additional 24 registers to a set (384 ÷ 16 = 24). It still takes four registers for a page frame in high memory. Under 4.0, however, that page frame can have a base address between 0000h and E000h, so that's actually 36 possible base addresses (or registers), not four.

If you had only a monochrome or CGA monitor, you could squeeze out an extra 64K of usable and swappable conventional memory at A000h through AFFFh. If you've got an IBM PS/2 model 50 or 60, you're going to have to swap right down to zero. The exact number varies, but to really implement 4.0 EMS and make it do the things that 4.0 promises, you need a bunch of registers, just to make a set. Because most everything in computers ultimately comes down to powers of two, 64 looks like a pretty good number (16 times as many registers as we needed under 3.2).

You could squeak by with 32 registers by limiting the floor to 256K (effectively ruling out use with several existing computers) and reducing the page frame choices to two. If you have less than that, you don't have full 4.0 support.

I stress this point because it is the main issue that separates boards built to minimum 3.2 support standards and boards that fully support all of 4.0's features. You cannot simply load a different driver and expect a 3.2-era board (other than EEMS) to give you 4.0 performance. The registers are just not there.

You need 64 registers to a set, but how many register sets? This answer is harder to quantify. Looking around the industry, AST's Rampage line provides 32 sets. The All ChargeCard, the memory management card that makes 286s almost act like 386s, adds no memory of its own but provides 16 register sets with 64 registers each to manage what is there.

The exact number of alternate register sets—hardware registers—is not as critical as the number of registers in a set. Unlike individual registers, additional sets of registers can be emulated, or faked, by storing the data that would be contained in RAM. Using DOS 5.0's 80386 EMS emulator, EMM386.EXE, emulated registers are what you get. There, however, is often a significant difference in performance. The reason is simple. An actual hardware register is a physical memory device like a RAM chip but dedicated to that specific task and always available to the EMM. If each register set is stored in its own dedicated device, then switching windows, for instance, is simply a matter of selecting an alternate register set that is in use and telling it to do its thing. All the pointers are in place, needing only to be activated. This set—virtually instantaneous.

In either case, however, the actual pointing must be done by a hardware device. If there is only one hardware register set, then every time you want to switch, two things have to happen. First, you must copy the contents of the real register into memory for storage (several output instructions, plus some other data, for each register). Then, you must copy the contents of another simulated register back into the real register every time you have to swap that 16K block out for another. Most programs today do the first step for you, so you can skip this step.

Because, under the old 3.2 specification, you were dealing with no more than four logical pages, you only needed four registers to a set. The time lost in copying register data was negligible, and you were only swapping data, not running code. With as many as 64 registers, this process is relatively slow going. Anytime you have to download and upload anything it's slower than just having it there at the ready. The difference here comes down to microseconds (millionths) versus milliseconds (thousandths) or a performance factor of about 10.

But what does all that really mean, you ask? Well, with enough hardware register sets to go around, AST boards have demonstrated the ability to do reliable background communications at 9600 baud. The AST boards were the only boards to achieve that level of performance in at least one widely publicized independent test. Not everyone has occasion to do background communications and certainly not at that speed except in special situations; however, this example is indicative of the board's potential.

I'm not saying that you necessarily need 32 or even 16 register sets now or for anything in the foreseeable future. Having at least some hardware alternates, however, certainly is desirable.

Having 64 registers in a set gets to the real heart and spirit of the 4.0 EMS specification. Under DOS, 64 registers are all that you can ever use. Remember that this whole issue of registers is unique to expanded memory. The degree to which they matter—if at all—depends entirely on the extent to which you will be using expanded memory.

The extent to which having hardware registers matters also depends on whether you are addressing memory directly as expanded memory at board level or are starting with extended memory and using part or all of it to *emulate* expanded memory, as is supported by DOS 5.0's new HIMEM.SYS and EMM386.EXE. At some point, you've got to have not only more sophisticated hardware but also more registers of some kind than were anticipated under the old 3.2 specification to enjoy all of 4.0's benefits.

Bus speed

Bus speed is another issue of importance, particularly when, after fully populating the motherboard, you have to start adding some sort of memory expansion boards to your system. With many of today's machines able to support as much as 8 Mb or more right on the motherboard, memory expansion boards cease to be a consideration on many of the newer machines. If it's on the motherboard (and you bought chips or SIMMs of the proper speed), memory runs at whatever speed the system is designed to run at. Good or bad, you're struck with it.

This issue is not specific to EMS 4.0 implementation by any means. It becomes an issue when choosing memory expansion boards for the faster machines that began appearing at about the same time. Not all chips or SIMMs will run at today's sizzling clock speeds. Many of the available memory expansion boards also are incapable of keeping up, no matter what chips or SIMMs you stuff them with.

Admittedly, few expansion boards are exposed to processor clock speeds but rather to bus speeds. Often, this speed is only half of the rated clock speed or less—even with accelerator cards installed. For an unmodified system, a 25 MHz box might have a 12.5 MHz bus speed. Many memory boards fall by the wayside with such speeds.

The bus in so-called state-of-the-art systems runs at something less than the dazzling clock speeds that the ads would lead us to suspect. Slower bus speeds—speeds compatible with the hardware that's out there on the bus—mean degraded performance.

Manufacturers are faced with a dilemma. They can slow the buses or they can just pull out all the stops and let them rip. The second option is not as irresponsible as it sounds because there are at least a few boards out there that can stand those withering speeds. Boards like Newer Technology's speed demons. All of Newer's memory boards boast proven compatibility with bus speeds to 12.5 MHz or higher with no wait states. Even Newer's Concentration board, which can be populated with up to a full 32 Mb of (soldered) chips on a single board, can handle that speed.

When all else fails, board designers can—and often do—add wait states. The designers don't always give the user much say in the matter, but sometimes they do. The attention! board from Newer Technology can keep up comfortably to bus speeds up to 12.5 MHz. Much above that, however, all bets are off.

The attention! has no less than eight software-selectable timing sets, ranging from none to multiple wait states. It has something for almost everyone. At the time of this writing, it had not been successfully matched to an NCR 916 Tower, but no other exceptions were known to exist. Using software to select the timing set is hardly the ideal solution; however, ours is not a perfect world. This setup works with a board that (with a piggyback) can add 16 Mb without overhanging the adjacent slot (1 Mb soldered SIMM chips are required to achieve this density). The board works well enough to have earned approval from SCO for use with XENIX, as well as by Novell and others.

These boards are not the only hot ones on the market, but they certainly are among the hottest. AST's boards generally stack up pretty well in most tests. As bus speeds go up, however, the list gets pretty short.

The bottom line

The hardware issues examined in this chapter are expanded memory issues and do not bear on extended memory, although it has become increasingly difficult—if not impossible—to totally separate the two. In either case, the system draws from a common pool of memory.

Despite their similarities, a paradox appears when you compare these two types of memory. Expanded memory is something that can be made available to even the most modest member of the 8086 clan. The effective delivery of full 4.0 EMS memory for multitasking and other high performance applications, however, requires a greater level of sophistication in the memory expansion boards and software drivers. You, therefore, need to weigh your intended memory use carefully when buying any new expansion boards to be sure the features you are buying best suit your actual needs and are not wasted.

6
CHAPTER

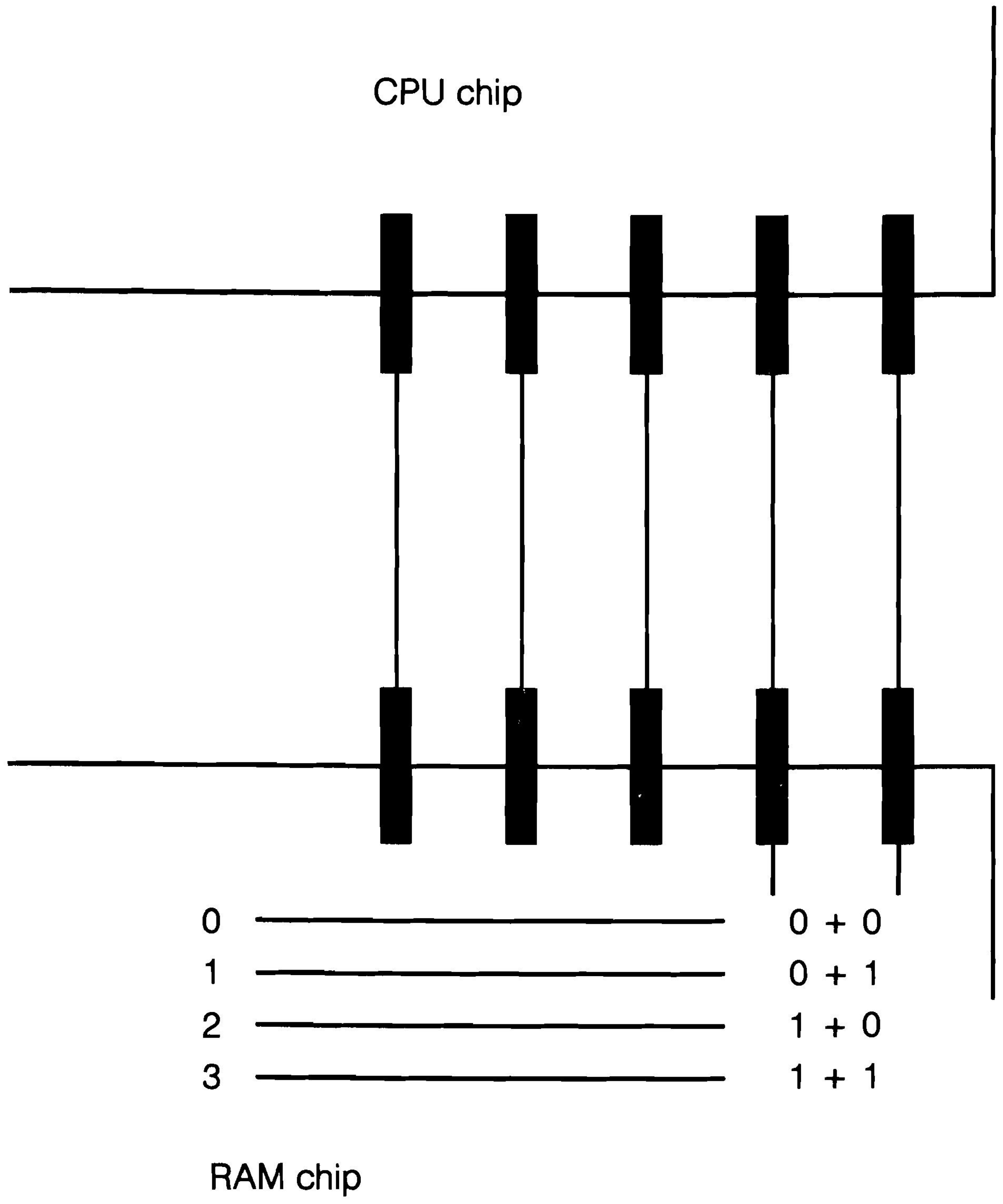
Stealing the store

The 80386 brought with it more than just the power to use extended memory as extended memory. The chip has an inherent ability to allow memory to be mapped to any unused DOS-addressable address, including mapping over areas that might contain system data that for one reason or another is not needed. The 80386 opened up a whole new area to DOS above 640K. DOS can use this area (200K or more) for TSRs, device drivers, etc., freeing the space they would have occupied in conventional memory for use by applications.

More recently, some of the software technology for memory mapping has trickled down to 286s and, to some extent, even to 8088s. (I'll look at those issues later in this chapter.) Essentially, mapped memory is the domain of 386 and higher systems. The rules don't change on lesser systems, but the numbers do—significantly. For the most part, I will deal specifically with 386 and higher systems and address those others later on.

The memory mapping allowed by the 80386 is not extended memory and has nothing to do with the high memory area (HMA). Although it is adjacent to the top of the mappable address range (actually overlapping it by about 16 bytes), the HMA is a totally separate issue. The memory mapping also is not expanded memory in the traditional sense, although the memory used must be drawn from an LIM 4.0 EMS pool. In much the same way as having a high memory area requires extended memory to supply that memory, you can have only memory that is mappable to addresses between 640K and 1024K (A000h to FFFFh).

Memory mapping is a term that's been around for awhile. Few users, however, understand just what it is or how it works. It's relatively easy to understand the concept of specific bits of memory having specific addresses when RAM is plugged into the motherboard—essentially hard-wired to address pins on the CPU. Figure 6-1 shows the relationship between specific address pins and specific addresses in low or conventional, memory.



6-1 Address pins have only two states: off or on. Each address represents the combined states off all of the address pins. Here, to simplify the illustration, I have concentrated on just two pins to demonstrate how these two pins control four addresses. A third pin would increase the number of available addresses to eight, a fourth to 16, and so on. Here the relationship between the CPU pins and the RAM chip address pins is fixed, the same pin's output is always directed to the exact same physical locations on the same RAM chip.

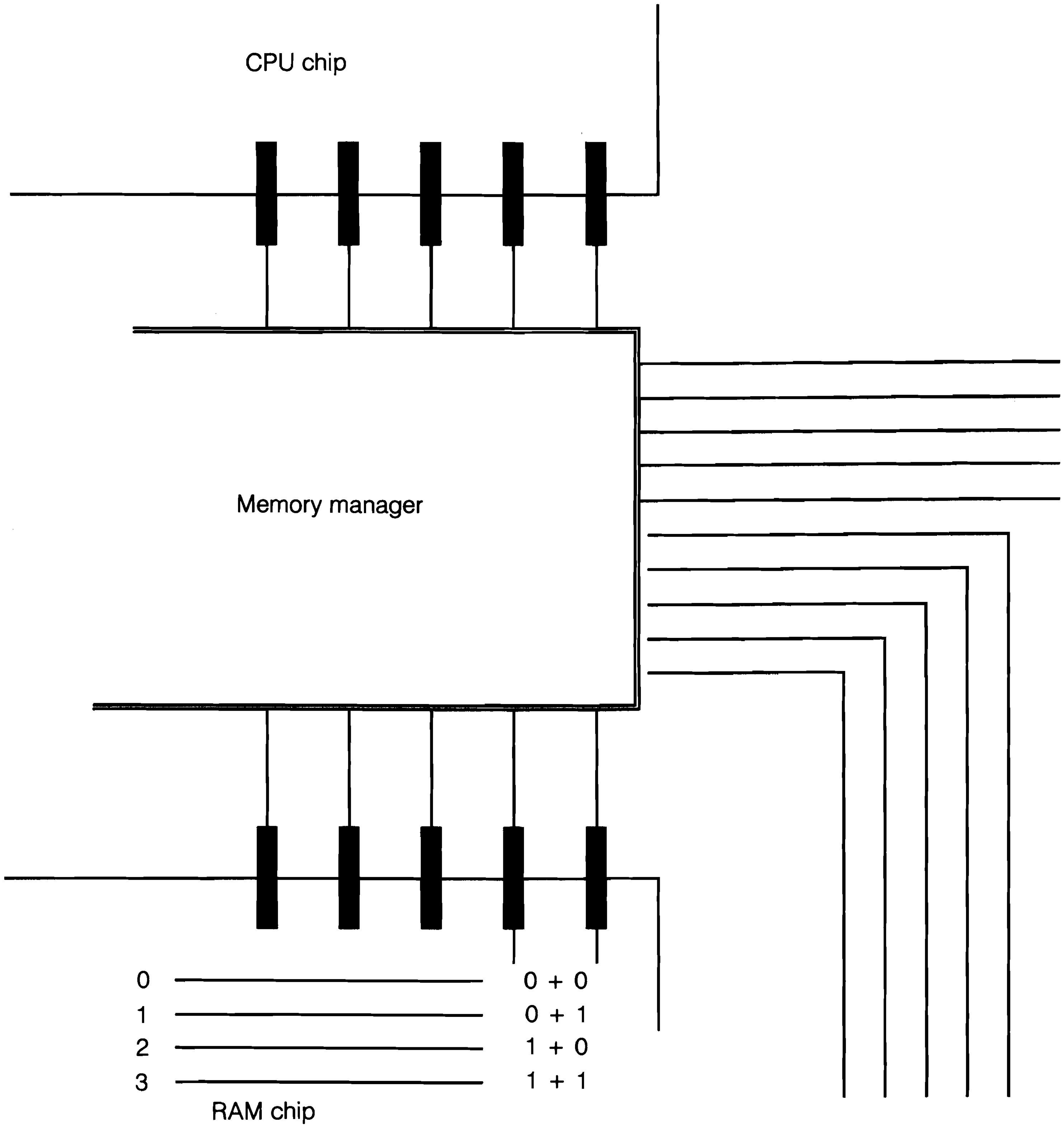
Somewhere, typically just above 640K, there is a block of addresses used for video. Those address pins, rather than going to user RAM, are normally reserved. They connect via the bus to the monitor card. As a general rule you can't map memory to any of those addresses because they're in use—or at least we assume they are.

In many cases, however, these addresses are not busy. In almost any character-based color video mode, the first 64K of the video area (A000h to B000h) is not used and can be mapped as additional conventional RAM (contiguous to 640K) by better memory managers, like QEMM or 386MAX.

Near the top of Fig. 6-1, there are some address pins that just dead-end. These are the empty addresses I compared to empty lots in a housing development in an earlier chapter. Empty lots have addresses, so do unused CPU address pins. In either case, you can send mail or messages. Just don't expect anyone to pick them up.

Figure 6-2 adds two more elements:

- Unassigned, or free, memory
- A memory management module



6-2 This illustration is similar to Fig. 6-1 except that the direct linkage between the CPU and RAM chip address pins has been broken and a memory manager inserted. The memory manager intercepts calls and can reroute them—like call forwarding with your telephone—to different locations. This is the key to accessing 4.0 EMS memory.

Assuming that the memory manager has already figured out which addresses are unused and available, it adopts those orphan address pins and acts something like a switchboard. The manager intercepts any calls sent to those addresses and redirects them to specific places somewhere in the pool of memory it owns.

A party of volunteers

On 386 systems, memory mapping is like an electronic muster. The EMS memory manager, the boss, looks the situation over and says something like "Okay, number 4C000h, you and your group report for duty at B400h." For the remainder of that work session, 4C000h will answer every call the CPU makes to B400h, 4C001h will answer for B401h, and so forth. These groups typically are 4096 bytes (4K), which is about the smallest block of unused (or reusable) addresses that can be worked with.

Throughout all of this mapping, DOS knows nothing about 4C000h and friends. DOS can't even count that high. Only the memory manager knows what's going on. DOS doesn't need to know as long as something, somewhere picks up and answers all the mail to those addresses.

This mapping of memory to high DOS (UMB) addresses differs significantly from the manner other EMS memory is managed. The difference being that, once mapped, those memory assignments generally remain fixed until the machine is powered down or rebooted. The physical memory addressed normally does not change as it did through the page frame that came with LIM 3.2 EMS. These assignments do not change even when bank switching entire applications and their data in and out, as occurs with multitasking. If the assignments did change, you would have to reload any device drivers or TSRs you might need for each application you were running. The solution is neither practical nor particularly workable. (Windows 3.0 does support a feature, which is currently supported by both QEMM and 386MAX, that does allow each window to have its own copy of customizing drivers such as ANSI.SYS.)

Once the assignments have been made (at the time the memory manager is installed from the CONFIG.SYS during bootup), they remain in effect until the system is rebooted. The memory manager, not the CPU, has jurisdication. The same actual physical bit of memory might be repeatedly assigned to make calls to the same address. If you change any value on the CONFIG.SYS line that loads the memory management device driver, the map determining who answers what also will change at the next reboot.

All of these changes are fine with the CPU; it doesn't care. Any old bit will do as long as the CPU can find the bit any time it needs the bit to respond to a call to some specific address within its megabyte.

This concept is really nothing new. It really is no different from the way an EMM must try to take control of every unused bit of memory in sight and dole it out. All that's really changed is that there's no longer just that 64K page frame to manage from the manager's revolving pool. There still is that memory; however, on a 386—and to some extent on many 286s—there also are the otherwise unusable addresses above 640K as well.

Managing all those other addresses does require some very different management techniques. Although there is a variety of device drivers available that can map memory, only the best can really make the most of your system's memory. To date, this area has been pretty much a seesaw battle between Quarterdeck and Qualitas, each fielding powerful, virtually unchallenged contenders. All—the people behind the ChargeCard for supercharging 286s—have jumped into the 386 memory management area with All Charge 386.

Now, even DOS has gotten into the act, introducing two new memory management tools with release 5.0. DOS also has included the ability to map memory to high DOS address space on 386s. Although they are not the most powerful memory management tools, the DOS tools are adequate for many purposes. Although they lack the frills of some of the more glitzy third party managers, the DOS tools demonstrate better than any of the better third party managers what is really involved in using upper memory.

DOS 5's HIMEM.SYS and company

No one is likely to ever call the memory managers that come with DOS 5.0 the greatest thing since indoor plumbing; however, they're not really all that bad. For our purposes, EMM386.EXE, the EMS emulator, is the one of greatest interest here. Understand that EMM386.EXE is not a totally free-standing program. Working from extended memory, an XMS extended memory manager such as HIMEM.SYS must be installed ahead of EMM386.EXE to provide that extended memory. Note that, despite its .EXE extension, EMM386.EXE is a device driver that must be installed from the CONFIG. SYS.

Full implementation of the memory mapping features of EMM386.EXE requires the support of the commands DEVICEHIGH (or DH) or LOADHIGH (or LH) that are internal to IO.SYS (one of MS-DOS's hidden files) in this release. As you can see, a number of factors actually come into play.

You need to write a CONFIG.SYS file based on using DOS's dynamic duo. In its barest form, your file would start something like this:

```
DEVICE=HIMEM.SYS
DEVICE=EMM386.EXE size RAM
DOS=UMB
```

This file is real bare bones, but it is a starting point. First, the extended memory (and HMA) manager prepares non-DOS memory and then the EMS emulator for use. When run at this stage, EMM386 reports something like the following:

```
EMM386 successfully installed
    Available expanded memory ..........     64   KB

    LIM/EMS version ...................    4.0
    Total expanded memory pages ........     28
    Available expanded memory pages .....     4
    Total handles .......................    64
    Active handles .....................      1
    Page frame segment ................   E000   H

    Total upper memory available ..........  75   KB
    Largest Upper Memory Block available ..  75   KB
    Upper memory starting address ........ CD00   H
EMM386 Active
```

You might have to put a hold on the boot process at this point (Ctrl–Num Lock) to keep the information from scrolling off screen too quickly. Although it is limited, there is some important information in this screen, especially the three lines near the bottom. They tell us that EMM386 has found just one block of what it considers to be mappable address space starting at CD00 and totaling 75K.

For reasons I'll discuss shortly, this amount doesn't necessarily mean that you can fit files totaling up to 75K into that space. This information also tells me that EMM386 has either not found or has ignored another 32K block I routinely use between B000h and B7FFh. If it is needed, there is a way to use it; however, for now, that 75K block will do quite nicely.

Although the EMM386 EMS emulator has a default size of 256K, you probably will want to specify some larger amount (in kilobytes) as the default size. Although it is more than adequate for mapping memory to high DOS address space, 256K is woefully inadequate for any serious expanded memory usage. The amount you set as the default size depends entirely on the needs of your individual system and the memory resources available. Finding the best balance might take a little trial and error. (If it is set too low, EMM386 will, under certain circumstances, override this setting if sufficient extended memory is available.) The RAM parameter in the following example specifies the use of mapped memory, while DOS=UMB enables memory to be mapped to high DOS addresses:

```
DEVICE=HIMEM.SYS
DEVICE=EMM386.EXE 2048 RAM
DOS=HIGH,UMB
DEVICEHIGH=RAMDRIVE.SYS 720 /E
DEVICEHIGH=SMARTDRV.SYS 360
DEVICEHIGH=MOUSE.SYS
DEVICEHIGH=EGA.SYS
DEVICEHIGH=STAK\STACKER.COM /B=C800 /M=0 D: \STACVOL.000
```

Now that you've got mapped memory to work with, you can start putting it to work. This point, however, is where DOS comes up short in the memory management department. You know how much memory has been mapped; however, how much memory is in use and what you can do with what is left—if anything—is another matter. All you can do is just throw programs at it—device drivers and TSRs—until you run out of memory. You'll get error messages when that happens. Nothing serious will happen; DOS will just load them low the way it would have anyway.

If you're not happy with which programs manage to fit and which ones get squeezed out, you can change the order that you try to load them, putting the more important ones (the biggest usually) closer to the top of the CONFIG.SYS file. Fitting the programs, however, is a real shot in the dark.

The DOS = HIGH statement in the above listing just tells DOS that it can have the high memory area and to load the kernel (47K at least) up there. DOS is not fussy about who's XMS driver you are using. This same command is valid for use with any XMS driver (if the DOS kernel is what you decide you want to load up there).

The MEM command, first introduced in DOS 4.0, will give you a little information about what's going on in high memory when using EMM386.EXE. Little of the information will be of use to most users as shown in this excerpt (which also shows some TSRs loaded from the AUTOEXEC.BAT or command line).

```
0CD010     IO                  00A1C0   System Data
             RAMDRIVE          0004A0     DEVICE=
               E:                           Installed Device Driver
             SMARTDRV          0045B0     DEVICE=
               E:SMARTAAR                   Installed Device Driver
             STACKER           005740     DEVICE=
               F:                           Installed Device Driver
0071E0     MSDOS               000030   -- Free --
0D7220     MODE                0001E0   Program
0D7410     XYKBD               000040   Environment
0D7460     XYKBD               0005D0   Program
0D7A40     MSDOS               0085B0   -- Free --
```

In this case, there was enough room for everything to load in upper memory. There, however, was one problem. One of the devices required a specific block of addresses that EMM386 had found and taken. You have to modify the command line to prevent EMM386 from using that block with the X parameter as shown below (the address range specified was detailed in the documentation for the device requiring that block be left open):

```
DEVICE=C:\DOS\EMM386.EXE 2048 RAM X=C800-CBFF
```

EMM386 also supports an I (Include) parameter that allows the user to force EMM386 to map areas it might not have found but that are thought to be usable—or reusable, as in the case of some ROM or video areas that can be mapped over. `I=B000-B7FF` tacked onto the DEVICE statement would have picked up that 32K block I mentioned earlier and mapped it too, but it also would introduce another problem that I'll discuss shortly.

If you work only with text-based applications, most display systems—CGA, EGA, and VGA—will allow you to add at least another 64K to conventional DOS memory with another I statement. (You can have multiple I and X statements and other parameters as long as you don't exceed the length of the DOS line.) I will discuss that option later in this chapter. Unless you're working in graphics mode, however, most of the area set aside for video usually is just wasted.

Any such exclusions and inclusions must be added manually with any of the memory managers. This limitation is not something DOS can be faulted for although some memory managers are certainly better than DOS at finding mappable blocks. Better third-party managers generally provide some sort of display of memory usage, making it easier to visualize how memory has been used.

Specific exclusions are something every user—even the beginner taking his or her first cautious steps above 640K—must be prepared to deal with. In most cases, you probably will find that the documentation for devices that might compete for address space will spell out exactly which blocks or space might be involved. In many cases, these programs will allow a choice to avoid conflicts with any other hardware that might be installed.

Includes should be approached more cautiously and probably are best avoided by beginners. Still, the worst that can happen is that you crash your system if you try to include an invalid block. (You always want to keep a bootable floppy handy at such times so you don't lock yourself out.) The best bet is to do as much experimentation as possible. Work from bootable floppies rather than your hard disk, changing that configuration only when you're pretty sure you've got the bugs out—most of them at least. Unfortunately, few managers will not allow you to install them to a floppy without a lot of aggravation. In those cases, having a bootable floppy with your old configuration close at hand is doubly important.

Actually, if you've got the patience, you can do quite well just throwing programs at upper memory blocks until you run out of space or develop a different loading strategy/sequence. I have managed to load as much as 146K (not including the use of the HMA and 64K EMS page frame), which probably figured out to something like one hour per kilobyte until I passed about the 96K mark and increased exponentially from there. There are a number of factors that can really complicate the process, not the least of which is trying to figure out just how big your programs are. There is often little correlation between file sizes and the space needed to load a program.

Bigger than life

FASTOPEN, weighing in with a filesize of something over 11K, really is a classic. In certain DOS releases, it has required as much as 68K of contiguous RAM to initialize, while in fact needing only 3K to run. This discrepancy borders on the obscene. There is no way you can—even with an accurate map of where you have been able to map memory to high DOS addresses—sit down and figure what you can, or should, load where without some careful scientific trial and error.

There are various strategies that can be applied to dealing with this kind of problem. You can load bullies like FASTOPEN first while there still is a fair amount of space to thrash around in, waiting until the dust settles to load your other programs (and in the process reusing some of that memory). At least one manager allows you to *borrow* from the EMS page frame, returning that address space to page frame use when the programs are finished loading. The All Charge 386 allows users to specify the starting address. You still might have to juggle your loading sequence in the CONFIG.SYS and AUTOEXEC.BAT files.

Auxiliary programs, such as Quarterdeck's Optimize for QEMM and Qualitas's Maximize for 386MAX, are able to perform wonders, eliminating most, if not all, of this trial and error for you. Even if these programs weren't a lot more powerful by nature than EMM386, this feature alone is worth the price of admission. With either of these premium third-party memory managers, you pretty much just have to install their software on your hard disk, execute the program, and then you're off and running.

These optimizing programs are not really smarter than most of us, but rather, they know exactly what to look for. The programs even know which device drivers and TSRs you have in your CONFIG.SYS and AUTOEXEC.BAT to try to relocate in upper memory.

Both Optimize and Maximize actually reboot your system several times before writing any changes to your CONFIG.SYS or AUTOEXEC.BAT files. The programs actually go through pretty much the same kind of trial-and-error process that you would, checking load and run sizes. The last reboot before they alter your files is to verify that the changes they are about to make will work. This process takes only minutes at most, not the hours it might take otherwise.

As smart, smooth and sophisticated as programs like Optimize and Maximize might be, the only thing they know how to do is find programs that can be relocated above 640K and load the programs there. Most of the time that's all that matters. Once in a while, however, they'll come up with a program that will load above 640K but just will not run right (or not run at all) because the programmer didn't take upper memory usage into account. This situation, however, cannot be anticipated. If, after using Maximize or Optimize, one of your pet TSRs or devices just won't run, you can remove any special loading instructions that might have been added to the lines that install them as shown here:

```
DEVICE=C:\DOS\QEMM\LOADHI.SYS /R:2
```

Fortunately, the number of programs that won't run properly when relocated to upper memory is relatively small these days and getting smaller.

Fragmentation

Further complicating the task of relocating TSRs and the like in upper memory is that, unlike the old familiar 640K, the memory mapped above 640K is often fragmented into bits and pieces. In the EMM386 example used earlier, there was only a single 75K block to work with. I said, however, that I knew of at least another 32K piece that was usable because I use it every day. There are several usable blocks (the minimum usable size is usually 4K) scattered around the high DOS area, including as much as 16K of ROM—often more—that can be mapped over if you know what you are doing.

If you pick up that 32K and another 16K and add it to our 75K, the total then comes to 123K. Any of the better memory managers will pick up some or all of it or we can select it manually with Includes. This 123K, however, is fragmented, with 75K still the largest single block available. As far as loading programs is concerned, 75K is the most available to any single program. The rest isn't even available even on loan to give a rambunctious program like FASTOPEN the extra elbow room it needs while loading because the memory is not contiguous.

If you look at memory fragmentation in the context of the overall picture, you can see how this whole mess began. DOS's megabyte was fragmented from day one when someone plopped the video at A000h, leaving users with a 640K fragment to work with. Now you're working with the crumbs, and the more memory you need, the smaller the pieces that you have to settle for. The rules are still the same. You can only load a program into contiguous memory unless the program itself specifically allows for fragmentation. (Since 1985, DESQview has been written to allow the use of discontiguous areas to maximize the use of available resources.)

I've compiled a partial list of programs that were actually loaded successfully above 640K on a working station:

18.0	Squish
20.0	Superpck
2.7	Packrmd
0.5	Mode
10.0	Map

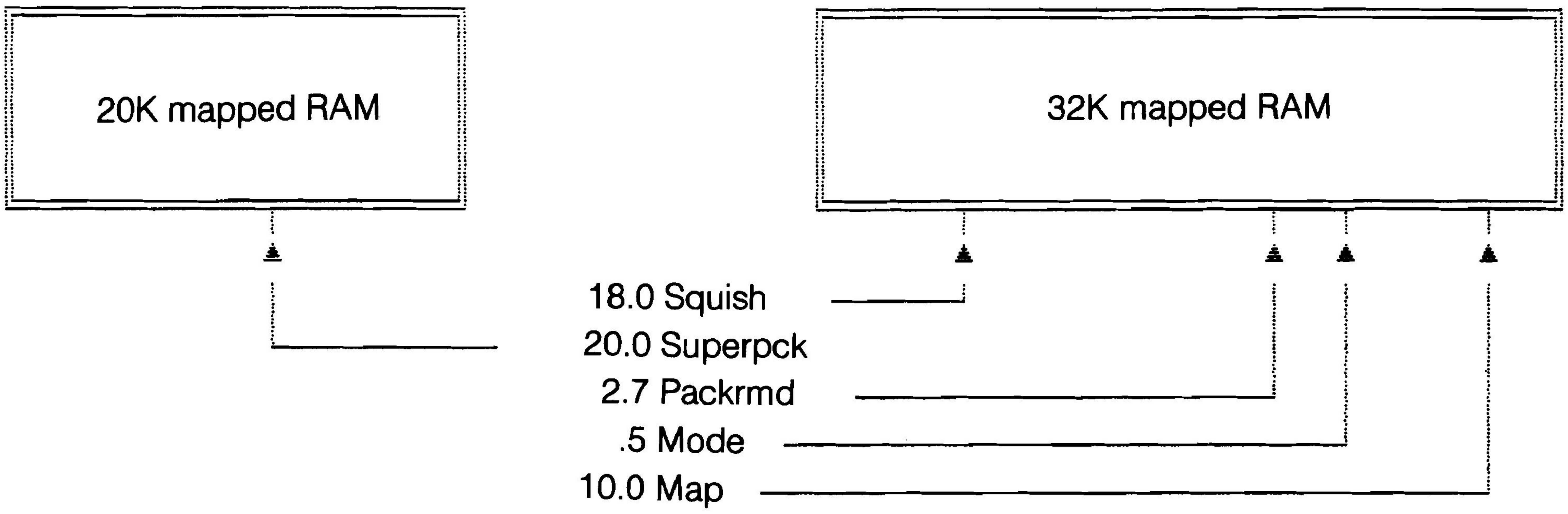
There are 51.2K of assorted drivers and TSRs listed—clearly more than you probably could afford space for in conventional memory. Assume for the moment that you have just 52K of recoverable address space in the high DOS area with RAM mapped to it.

Loading the program would be easy except that the high memory is fragmented. One 20K block is squeezed between VGA graphics and text, the other 32K is just underneath the 64K EMS page frame. Although I have changed the size of the blocks to keep the example fairly obvious and simple, these particular locations correlate with high memory blocks in an actual system.

They should all fit, but here is what happens. Three of the five files listed are device drivers loaded from the CONFIG.SYS (PACKRAMD.SYS, MAP.SYS and SQPLUS.SYS). PACKRAMD, a RAM disk driver, was already in use when the other files were added to the system, so they were just tacked on to the end of the CONFIG.SYS. These other programs are going to settle in before the system even looks to the AUTOEXEC file for whatever else is to be loaded up there, including that 20K piece from SUPERPCK.COM that has to be loaded somewhere for PACKRAMD to work.

Even at a glance, you can see that, if that 2.7K loads first and picks its spot in the lower 20K block, there won't be room there for either of those 18K to 20K chunks. At least one of them must load into that 20K for everything to fit.

Juggling things so the 18K loads first into that lower block doesn't do any good. MODE is the only other program that could fit with it. The remaining 32.7K has to try to squeeze into a 32K parking space. It just isn't going to fit. If you just let nature take its course, there probably wouldn't be space to fit that 20K piece up high at all. As you can see from Fig. 6-3, even with HMA space to spare, you easily could wind up wasting 20K of conventional memory.



6-3 The loading order of blocks can affect how the programs will fit (or not fit) in memory.

The only way to make all the programs fit is if you can somehow save that 20K block for the late-loading 20K file. In the real world, the more fragmented upper memory is and the more puzzle pieces you have, the more complicated the task. I cannot stress this point too strongly. Only when you understand the issues can you be sure the memory manager you spend your money on is up to doing all that you expect of it.

Even this example assumes a lot of things you just can't take for granted—like the possibility that one of the programs might require more elbow room while loading than is reflected here. As the crumbs get smaller, care must be taken not to waste big spaces on a lot of little programs that could as easily fit in smaller pieces. Programs like Maximize and Optimize take this consideration into account automatically when they work out your loading sequence and strategy. You can determine the loading sequence and strategy manually, with a lot of trial and error. If you value your time as worth anything, however, be sure the memory manager you buy can do this process for you.

Declaring open season on the BIOS

Traditionally, the 64K occupied by the ROM BIOS (128K in the case of all of the PS/2 series from IBM) has been considered to be off-limits. There is a general feeling among most users that any fiddling around up there is sure to crash the system. Indiscriminate fiddling surely can and will; however, some of it can be mapped over and reused.

Much of the ROM space is empty or occupied by unused or unnecessary functions on almost all machines. (Sometimes when you aren't doing anything, go poke around up there a little with DEBUG and see how much empty space there is.) Most of the space is in blocks too small to be of much real use unfortunately; however, much that is in use can be written over without ill effects. Deciding which parts you can write over safely is something better left to experts, but the space is there.

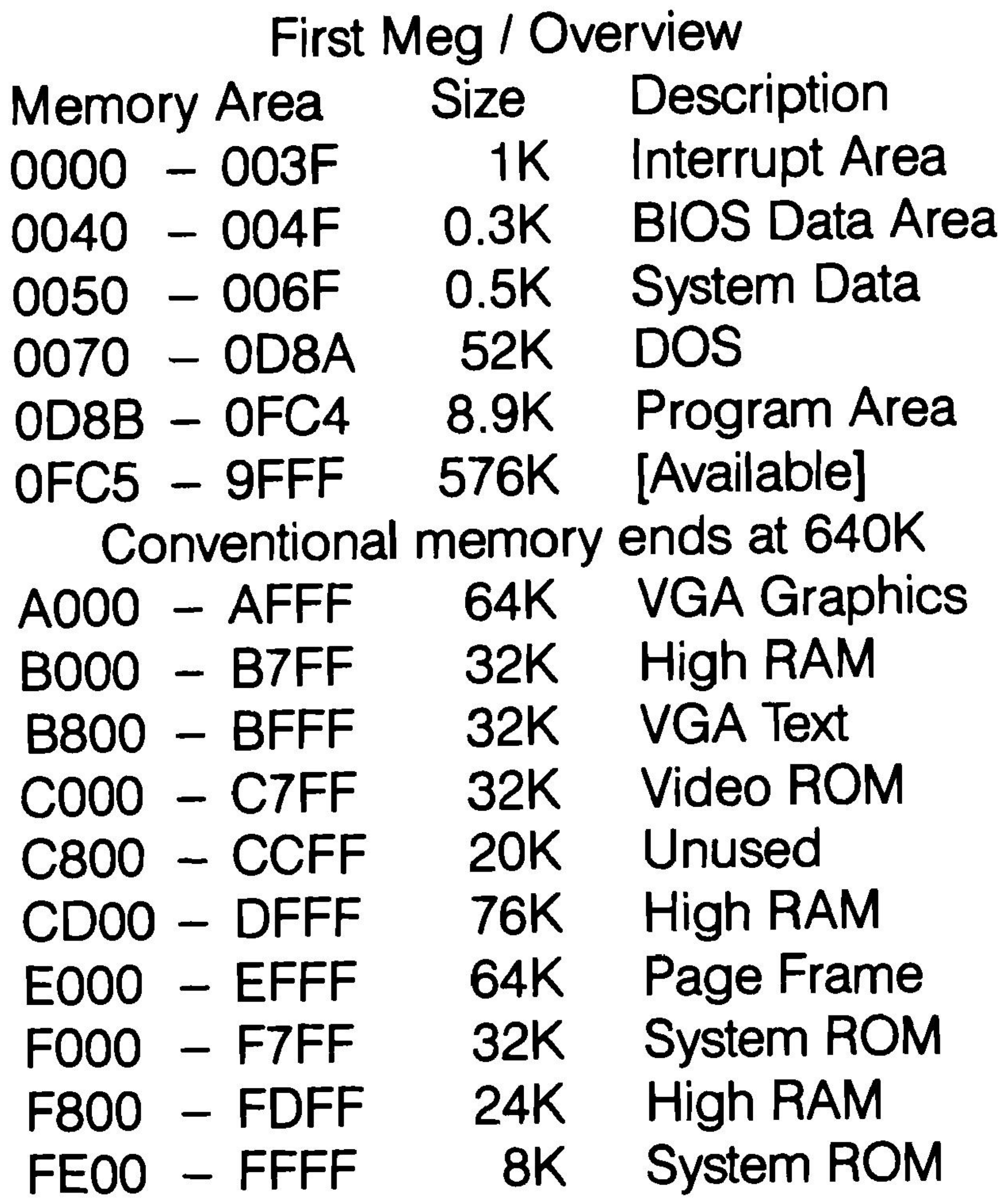
Quarterdeck's QEMM386.SYS is smart enough to sniff blocks of ROM address space it can appropriate. On the system that I wrote most of this book on, QEMM mapped more than 24K of ROM space, which, although representing more than one-third of the total ROM area on that machine, is good but is not remarkable because the amount that can be mapped over is as high as 40K on some machines.

QEMM can reclaim as much as 24K out of the BIOS region—possibly even more on some systems, as shown in Fig. 6-4. This isn't space that ROM doesn't occupy but rather is ROM that provides no essential services—at least not once the system is up and running, which is all that really matters.

Except for BlueMAX, Qualitas normally doesn't tamper with ROM space. It, however, does allow you to Include any you think you can get—as do most other managers. It's pretty picky business though, but then this whole upper memory area is.

For want of a map

For want of a map, the byte was lost; for want of the byte . . . . Unfortunately, even if you had access to the latest versions of all the better memory managers, figuring out which one was best for your particular system isn't easy, except per-



```
               First Meg / Overview
Memory Area        Size    Description
0000  - 003F         1K    Interrupt Area
0040  - 004F       0.3K    BIOS Data Area
0050  - 006F       0.5K    System Data
0070  - 0D8A        52K    DOS
0D8B - 0FC4        8.9K    Program Area
0FC5 - 9FFF        576K    [Available]
      Conventional memory ends at 640K
A000  - AFFF        64K    VGA Graphics
B000  - B7FF        32K    High RAM
B800  - BFFF        32K    VGA Text
C000  - C7FF        32K    Video ROM
C800  - CCFF        20K    Unused
CD00 - DFFF         76K    High RAM
E000  - EFFF        64K    Page Frame
F000  - F7FF        32K    System ROM
F800  - FDFF        24K    High RAM
FE00  - FFFF         8K    System ROM
```

6-4 Much of ROM can be mapped over on almost any machine that supports mapping. Here, without any help, QEMM has found 24K (F800h to FDFF) that can be put to better use.

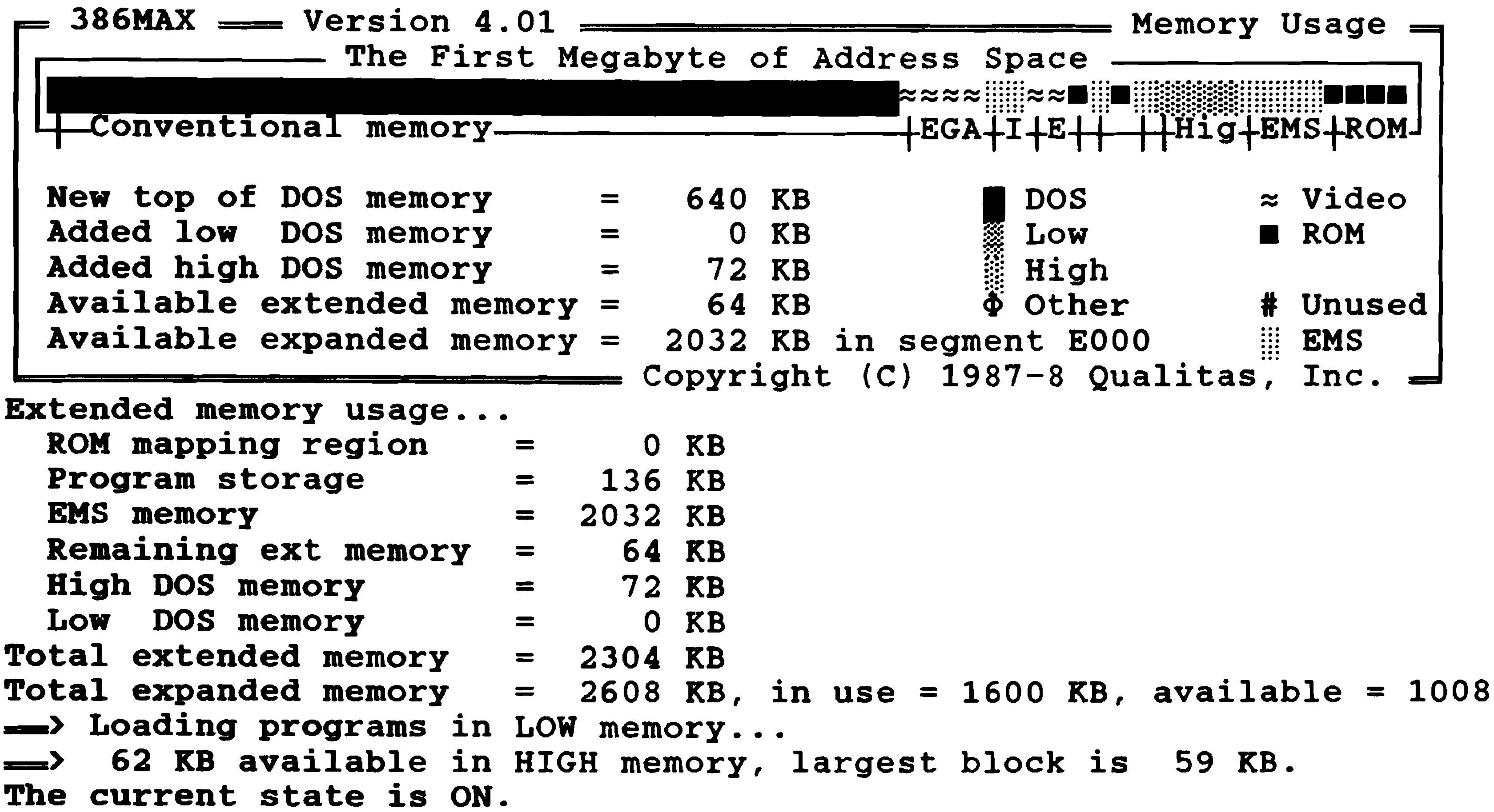
haps in the case of some of the super specialists. One of the greatest frustrations is that there is, as of this writing, no comprehensive universal mapping utility.

At best, some of the better management packages provide their own proprietary mapping programs aimed at demonstrating how good a job the home team does, but that is all they do. Don't try to check to see how 386MAX does with QEMM's Manifest, or anybody else's clever demo map.

Unfortunately, no one has marketed a memory manager that is compatible with the competition. It's not just proprietary nastiness, however. Each of the developers has his or her own individual way of staking out a claim in high memory. 386MAX not only leaves its own distinctive signature marking the point where it starts loading anything in high DOS memory but also leaves its mark on any unused high memory.

Other managers are more subtle than 386MAX. In an ever changing field, it is impossible for anyone to keep up with all the tricks that developers are using, let alone figure out a way to show them for comparison—even if they wanted to. Therefore, there are a bunch of different memory and system use displays from different vendors. You already have seen what DOS provides, which isn't very much. DR-DOS 5.0 does better in several ways.

Quarterdeck's MANIFEST is certainly the glitziest of the bunch and overall probably presents the most useful information. Its presentations, however, are entirely different than those of Qualitas's 386MAX, shown in Fig. 6-5 (a .COM file with the same name as the Memory Manager) or All Computer's ALLMENU. I have tried here to get the best possible comparative look at the performance of each of the three memory managers these represent, but as you can see, considerable interpretation is involved.



```
╒ 386MAX ══ Version 4.01 ═══════════════════════ Memory Usage ╕
│┌──────── The First Megabyte of Address Space ──────────────┐│
││█████████████████████████████████≈≈≈≈░░≈≈■░■░░▒▒▒░░░■■■■││
│└┼Conventional memory────────────┼EGA┼I┼E┼┼──┼┼Hig┼EMS┼ROM┘│
│                                                              │
│ New top of DOS memory     =   640 KB       █ DOS     ≈ Video │
│ Added low  DOS memory     =     0 KB       ▒ Low     ■ ROM   │
│ Added high DOS memory     =    72 KB       ░ High            │
│ Available extended memory =    64 KB       Φ Other   # Unused│
│ Available expanded memory =  2032 KB in segment E000 ░ EMS   │
╘══════════════════════ Copyright (C) 1987-8 Qualitas, Inc. ══╛
Extended memory usage...
  ROM mapping region      =     0 KB
  Program storage         =   136 KB
  EMS memory              =  2032 KB
  Remaining ext memory    =    64 KB
  High DOS memory         =    72 KB
  Low  DOS memory         =     0 KB
Total extended memory     =  2304 KB
Total expanded memory     =  2608 KB, in use = 1600 KB, available = 1008
══> Loading programs in LOW memory...
══>  62 KB available in HIGH memory, largest block is  59 KB.
The current state is ON.
```

6-5 Quarterdeck's Manifest is clearly the best utility around for system memory usage analysis and system informatio general. Generally bundled with Quarterdeck products, it also is available separately and can be used on machine from the 8088 on up, regardless of whether other Quarterdeck software is present. Because of propriet differences, Manifest cannot give full information on the performance of memory management products other t products from Quarterdeck.

In these three illustrations, there are discrepancies even in the amount of memory shown as being used by the same TSRs or device drivers. This variation, however, might not be an error in reporting, but rather reflect differences in the amount of memory actually allocated by different managers. Some TSRs and drivers actually do require different amounts of memory at different times. This problem further complicates the situation and will be discussed more fully elsewhere in this book.

Some third-party utilities are available that can be of some help, notably System Sleuth, which was used in making one of the preceding illustrations, and InfoSpotter. Both utilities are relative newcomers on the scene. These utilities have no ax to grind and, within their capabilities, the information they present is unbiased. Because different software developers leave different kinds of signatures on the memory map there is no way of seeing what is going on—at least so far. The serious worker, therefore, is still hard-pressed to get the information necessary to squeeze the most out of the high DOS area.

It went where?

The whole purpose of this exercise is to increase the amount of conventional memory available to run your applications software. The first few chunks usually come pretty easy. If you pursue the quest beyond those first few easy pieces, however, you can run into a situation where the more you succeed in moving things up high, the less you have to show for it. I don't mean less in the literal sense (hopefully) but in the sense that moving one or more additional TSRs up high might not give you even one more byte of memory—not even the space the TSRs occupied down low.

The same forces were at work from the time you relocated your first TSR. In the heady euphoria, however, you probably would not have noticed if the 23K you moved only gave you 16K of benefit. The deal probably is still a good one, but where'd the other memory go?

Numbers lie. Actually, in this case, it isn't the numbers that lie. The rules in upper memory are different. Earlier, I indicated that 4K was the smallest block size usable in upper memory for most purposes. If you have six contiguous 4K blocks, assuming no loading peculiarities, that amount will be enough to hold the 23K program.

Down in conventional memory, however, you don't work with 4K blocks. 16K is a more common block size down there, but then, when you're reduced to eating crumbs, you have to lower standards. Programs can overlap those 16K boundaries and sometimes even share, so depending on where the break point falls, you actually might free only one 16K block. On the other hand, you might free two and wind up gaining 32K of usable memory. In one extraordinary case, I set off a 32K avalanche by moving only 9K.

Even knowing this principle, trying to free up even one more block to use down low sometimes can get really frustrating. In any case, you need to remember that what you load up high is what counts. What matters is how big a chunk you've got down low to load your applications. If you ever get down to having to make a choice because you can't fit everything upstairs, go for the combination that leaves the most for you to use down low because that's what it's all about.

Stealing still another 64K—maybe even 96K

Users working only with character-based applications usually can add an extra 64K to as much as 96K to conventional memory by using still one more trick that some of the better memory managers have up their sleeves. This technique is not limited just to the 386s either. It can be done with 286s and even 8088s.

The truth is that many—if not most of us —actually are wasting most of the memory normally reserved for video use. For character-based screens you actually need only a few kilobytes, the remainder is needed only when you venture off into some graphical environment.

The little memory that the video system needs, even with EGAs and VGAs, is somewhere in the B region, leaving at least the entire A000h to AFFFh 64K segment—contiguous to the unused 640K. You'll recall from the earlier discussion that any block of free memory that's contiguous to the 640K can be added directly to conventional memory. Only when you break the continuity does DOS have to put on the brakes. DOS normally can't use this space for anything but video graphics. With mappable memory, this facet of memory management is mainly just a matter of taking down the signs and letting DOS move in—right on up to B7FFh with an EGA or VGA, for a whopping 96K gain.

This memory is additional conventional memory for running bigger applications and bigger spreadsheets or for loading the entire manuscript for a book like this into RAM in one gulp, along with your word processor. You still have the HMA and every byte of memory you've mapped as upper memory blocks above the video.

This memory is the same bonus memory I mentioned in an early chapter in conjunction with using CGA displays. In that context, a few vendors made it available to users; most didn't. Then with EGAs and VGAs, everyone pretty much forgot about it. If you treat your EGAs and VGAs like CGAs, however, the extra RAM is still there but with the vastly better screen resolution you paid for when you bought your better monitors. In this way, you can have your cake and eat it, too.

There are a few programs that seem to take exception to this incursion for unaccountable reasons. The BIOS used by some computers also makes assumptions about the use of the A000h to AFFFh area that make it impossible to use this region. Fortunately, most programs and BIOSs could care less. This information, however, is something to file away in the back of your mind in case at some point your display suddenly goes bonkers when you try to load something.

QEMM386 users can grab this unused video RAM by using the utility, VIDRAM.COM, which can be loaded into memory mapped to addresses above the video if you've still got any left. VIDRAM can be toggled on and off in mid session if you wish—to run a desktop publishing or windows application, for example.

There is one catch, though. You cannot toggle VIDRAM from within DESQview or any windowing environment. You have to exit the environment to toggle VIDRAM, even if you want to load it right back up again. This inconvenience, however, is a small price to pay. In practice, you are not likely to want to toggle VIDRAM except when changing environments, anyway.

Both Qualtas and All have provisions that allow users to include this portion of the video area in the space available to DOS. All's ALLMEN4 driver will take the memory by default if the drive detects a CGA graphics card, rather than an EGA or VGA. With either of these boards, you must set a parameter in the CONFIG.SYS. In any case, 64K is 64K. If you're not into graphics, go for the extra memory, especially if you've got a 286.

Don't count the 80286 out yet

Qualitas mapped high memory first it seems, mapping LIM 4.0 EMS memory to high DOS addresses on an 80286 with a memory manager called MOVE'EM. Quarterdeck, however, was hot on Qualitas' heels with QRAM (pronounced cram). In 386s, the CPU chip makes the difference more than the support chips on the board. With a 286 or 8088, the difference is in the supporting cast, as demonstrated earlier by All Computer's ChargeCard (discussed in detail in chapter 16). The Chips and Technologies chip set, used on some of the better 286s, seems to give about the best performance short of going the ChargeCard route.

In the best of circumstances, neither of these software solutions—and probably any that might follow—can match the near-386 power and performance of a ChargeCard-upgraded 286. These software-only packages, however, are not nearly as expensive as the ChargeCard board, selling at around $90.

Even the vendors will tell you frankly that performance will vary considerably. Quarterdeck claims 30K to about 130K for its package. I squeezed out an extra 64,096 bytes with MOVE'EM on an AST Bravo 286 with an EGA. This amount is in addition to the 1,408 bytes for MOVE'EM's own overhead. Higher numbers with better chip sets, therefore, would certainly seem reasonable. Here again if you're not running in graphics mode with 4.0 EMS memory to map and one of these to pull it off, you can probably add at least an extra 64K to your conventional DOS area, bringing you up to 704K and, in some cases, even 736K of contiguous DOS memory.

I have devoted a rather disproportionate amount of time and space here to an area that is relatively small at best. The difference that mapped memory can make, however, is no small matter and, as you can see, is the one memory area where understanding what you're doing can really make a difference.

7

CHAPTER

Extended memory and new frontiers

If expanded memory is like a lazy Susan, then extended memory is a little like a hundred-story elevator in a 10 story building. It goes right up through the roof and keeps going. It had better not be gone, however, because, if you can't get back down to DOS where you started from, you're dead. Getting back to DOS is where the 80286 ran into trouble. It would let you go, but it would fight you coming back. Everyone had to wait for the 80386 to really see extended memory work, but it was worth the wait.

Extended memory is the good stuff—continuous linear memory. It is not only bigger and better but far faster than expanded memory, particularly on 32-bit systems. To the CPU, extended memory looks like an unbroken string, with addresses that start at 1024K and keep going until your CPU runs out of address pins or you run out of money—more likely the latter. With OS/2 and probably most other operating systems that you're likely to encounter, all the memory you have is just one big happy family, just one long string of addresses without distinction. With DOS, this last outpost is called extended memory, at least for now.

Promised with the introduction of the 80286 (a promise that the chip could never quite fulfill), extended memory was a long time coming. With the rapid acceptance of the 80386 and its SX and SL counterparts and now with the emerging 486 market, extended memory is coming on as the most dynamic area of expansion, not just for tomorrow but for today. With new software and the power it unlocks even on today's machines, extended memory is one of the reasons microcomputer sales have already eclipsed mainframe sales by something like 65%. In the rush to tap the full capabilities of the 386 and 486 computer chips—

and the failure of OS/2 to emerge as a viable operating system—many software developers have taken a new look at the 80286, too.

Extended memory is contiguous, as opposed to expanded's lazy Susan now-you-see-it, now-you-don't revolving access. They are two completely different systems, requiring different software access.

To another operating system designed without the 1 Mb constraints of DOS, extended memory would be ordinary conventional memory. You can have up to 4 gigabytes of continuous memory on a full 80386 DX or *i*486 with a true 32-bit operating system (accessed in protected mode above 1 Mb), but only 16 Mb on a 32-bit SX.

Using spinoffs of that technology, some developers have taken new looks at the poor old 80286 as well, because it also has the capability of handling a full 16 Mb of extended memory (plus 32 megs of EMS expanded memory). Even without the benefit of a true 16-bit operating system, like the still elusive OS/2, extender technology has paved the way for powerful 16-bit processing from DOS. This raw processing power is unleashed for running applications, even huge applications, at full throttle right from DOS.

Still, the difference between extended and expanded memory continues to be one of the greatest sources of confusion and consternation, both to users and programmers alike, although for rather different reasons. Users find it difficult to understand the difference between the two. Programmers found that for a long time extended memory proved unruly and difficult, if not at times impossible, to work with in the DOS environment.

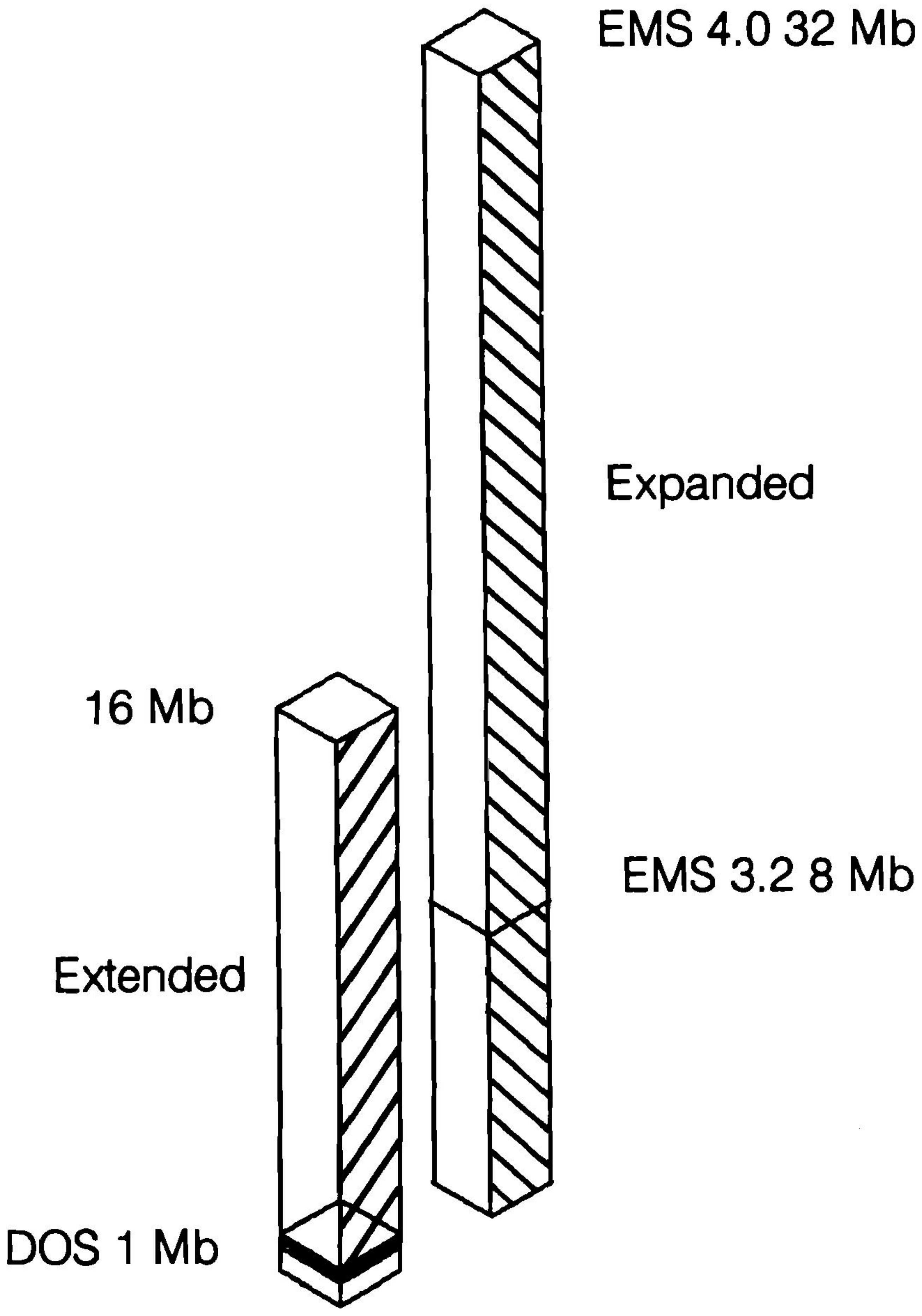
A quick review

As shown on the left side of Fig. 7-1, extended memory (sometimes aptly called linear memory) is simply a linear continuation of addresses beyond 1 Mb (10000h), as compared to the bank-switched blocks accessed sideways, as it were, through discreet 16K pages, as depicted on the right. Note that there are only 16 Mb available under extended memory, while expanded memory gives access to as much as 32 Mb under the LIM 4.0 EMS specification.

The address pin limitation discussed in chapter 1 prohibits adding memory beyond those points. The same thing limits the 8088—and DOS—to only 1 Mb. For either an 80286 or 386 SX, the maximum extended memory that can be addressed is 16 Mb (24 address pins or 2^{24}). The numbers for the 80386 quickly become seemingly unreal, so for the moment, I'll confine discussion just to the 16 Mb extended memory limit of the SX or 286, which is only half the memory limit available using any microprocessor chip under DOS.

The critical fact to sort out in your mind at this point—if it isn't sorted out already—is that there are two totally different ways of accessing memory beyond 640K. Expanded memory is always broken into 16K pages. By stringing pages

7-1 It is important to have the distinction between extended and expanded memory firmly in mind. Here I have depicted the memory scheme for an 80286 where the maximums available can be depicted on the same scale.



together, you can do a lot of clever things with it; however, the bigger the program the more difficult it becomes. But even in a best-case situation with full 4.0 EMS support, as soon as you exceed the size of your conventional DOS memory, you have to start juggling pages in and out. The more you have to juggle, the more time you waste.

Extended memory knows no such limitation, requires no bank switching, and is bound only by the address limitations of the CPU (and your pocket book) right up to the 386's 4 gigabyte (2^{30}) extremity. That's 250 times as much.

Because 16 Mb is surely more than adequate for most users, logical questions arise. Why should you even bother with all this hocus-pocus of logical pages and frames and lazy Susans? Why not just buy a 286 or 386 (if you don't already have one) or an accelerator card for your old PC and just go for it?

It's not that easy. First, you've got to go into protected mode.

Protected mode

More people say more things and know less about protected mode than almost anything, except perhaps the weather. What protected mode is, in concept, really is rather simple. Beginning with the 80286, Intel's designers tried to implement a scheme that would protect virtual addresses (in DOS terms, linear addresses beyond 1 Mb) so that multiple operations could be run there concurrently. The integrity of each address and its attendant data was protected from the others.

The most significant difference between running in real mode and in protected mode is that, in protected mode, segment registers contain *selectors* rather than actual physical addresses. This difference is critical. You need to understand it.

Selectors are a lot like substitution tables. A call directed to any specific segment or address is intercepted and rerouted via a selector to the place that code or data is actually located, which the selector knows because it put the data there in the first place. The process is sort of like having an appointment with the President. First, the Secret Service has to look you over, then maybe they'll tell you where he is.

These selectors provided an extra level of indirection when accessing memory. Instead of being the base address of the segment in memory, a selector is merely an offset into a table of *descriptors*.

Each descriptor contains the base address and length of the segment, as well as additional information required to implement the memory protection features of protected mode. The values loaded into the segment registers do not correspond directly to physical addresses.

For an application to access a particular physical address (like screen memory), it must first load the base address of that area of memory into a descriptor and load the selector that corresponds to that descriptor into a segment register. Two tables of descriptors are available to each process. One table is called the Local Descriptor Table (LDT), and the other is called the Global Descriptor Table (GDT). The GDT is shared by all processes in the system, but each process has its own LDT. The GDT normally maps system-wide data structures and the LDT maps process-specific data structures.

The use of descriptors requires a chip with an architecture that, unlike the 8086 and 8088, has the internal capability to do this. It is relatively simple, and logical. You need to have something like that going for you to prevent utter chaos outside the relatively well ordered but narrow realm of DOS.

However, it didn't work, not with the 80286 at any rate—at least not well. In hindsight, someone apparently dropped the ball rather badly when the 286 chip was designed. To this day, you hear cries of "foul" and see fingers pointed in various directions. Whatever the underlying cause, the bottom line is that with an 80286 you cannot slip easily back and forth between real mode and the protected mode without crashing the system or walking a fine line on the brink of doom. (In the real mode, an 80286 chip or higher behaves just like the original 8086 with the original 640K/1 Mb limitations.) This problem is attributed by some experts to a conflict between DOS's own internal access needs and the internal addresses needed for the instructions necessary to shift the 80286 in and out of the protected mode.

A number of important design differences in the 80386 eliminated those problems. Protected mode is now a practical reality that programmers can work with. The possibilities are hard to even comprehend—complex applications possibly as big as 15 Mb. Some implementations of OS/2 apparently might use this ability to allow real mode PC/MS-DOS applications to run concurrently (multitasking) with protected mode operations running under OS/2 (or some other protected mode operating system).

Even after the 80386 was introduced, there were still obstacles to overcome before extended memory could come into its own. Like any frontier area, there were disputes, even about how it should be accessed.

Bottoms up

In the absence of a standard—or even a concensus—for how extended memory should be accessed during the first three or four years after the introduction of the AT, software developers were left pretty much to do their own thing up there. It was a lawless place. There were no good guys and no bad guys really. There were just a few guys scratching out a living up there, or trying to at least. Driven up there from a world of starving applications, they scratched and nibbled at extended memory, mostly just around the edges.

At first, there was not much available besides RAM disks, print spoolers, disk caching schemes. Soon, however, other people started looking at extended memory. AutoCAD and several others started stuffing data up there. There still was no law and order. If you tried to put two tasks, two sets of data, or two anythings up there, both usually would crash.

During this time, two quite different philosophies evolved. IBM's VDISK, which, beginning with DOS 3.0, has been capable of using either conventional or extended memory, started at the bottom of the pile and ate its way up toward the top. The other school of thought said extended memory should be accessed from the top down. There were a number of logical arguments to back up this opinion, one of the more powerful arguments is that it is less complicated and easier to see who else is running up there (providing everybody else also is working from the top down, which they weren't) and how much space they're occupying.

Everyone knows what happens if you try to burn a candle at both ends: messy drips all over everything. The usage of extended memory got really messy really fast. If you tried to run more than one program at a time up there, someone's data usually wound up dripping all over someone else's data and you had corrupted files and all kinds of nastiness.

The situation was further complicated by several noncooperating applications that assumed that any extended memory present when they were running was their exclusive property. The programs went into protected mode and started writing directly to the memory, without bothering to check for other users.

Not surprisingly, extended memory got a lot of bad press in the beginning—its reputation was made even worse by the grotesque gyrations required to bring an 80286 back down to earth of real mode once you got up there.

In 1988, Microsoft, in collaboration with AST, Intel, and Lotus, released the eXtended Memory Specification (XMS) which defined an interface comparable to the role played by the LIM EMS standard for expanded memory. That paper defines what is now the industry standard interface for allowing real mode programs to access and use extended memory on machines with 80286 processors and higher. It, however, goes much farther than that. Overall, it defines a set of rules governing:

- The use of upper memory blocks (UMBs) between 640K and 1024K—the area where memory can be mapped into to relocate device drivers and TSRs from conventional memory
- The High Memory Area (HMA) between 1024K and 1088K—the extra 64K of DOS-addressable memory available to 386s and higher machines
- The use of extended memory blocks at addresses on up as far as you've got bucks to buy the chips for, basically

The XMS also defines a hardware-independent mechanism for controlling the A20 gate. The gate must be opened every time control passes back and forth between real and protected modes—as when a program running in extended memory must return to real mode for keyboard input, disk I/O, etc.

Quarterdeck had done much of the pioneering work in both the UMB and HMA areas. Although much of the company's work was seemingly incorporated in the new specification, it was not a party to drafting the specification.

Enter the DOS extender

The promise of a true 16-bit operating system for the ill-starred 286 still remained unfulfilled when the 80386 burst on the scene with a protected mode that really worked and with full 32-bit processing power. Much of the incentive faded for a 16-bit OS/2. There was and still is talk of something else, a 32-bit OS/3 perhaps.

New operating systems, however, aren't born overnight. OS/2 still is proving that. The 16-bit OS/2 still was not available when 32-bit power—really fast 80386 ATs, but little more—was already sitting, waiting on people's desks.

In this vacuum, several software developers began to look for some way to work from DOS. Essentially, they wanted to use the available DOS as a launch platform, while doing the actual processing out in extended memory. The logic was impeccable. The idea didn't need a whole new operating system, just a superset for DOS—a 16-bit instruction set that could run on 16-bit or 32-bit machines, or a super 32-bit set for 80386 and *i*486 machines.

With rumbles even about the possibility of 64-bit chips to come, the superset idea had merit. Because it did not require the lengthy process of writing a whole new operating system, the superset could be done quickly and in ways that would allow existing applications to be ported to this modified environment faster and more economically than might be possible than to some all-new operating system. And so the DOS extender had arrived.

A DOS extender is a mini operating system that loads on top of DOS, picking up where DOS leaves off. The way extenders operate is entirely different than expanded memory managers. (Extenders, for instance, are generally not installable devices but rather merely part of a program loadable from the DOS prompt.) Both, however, are in their own way parallel in that each provides important services lacking in the underlying operating system.

A DOS extender also must provide a way to interface with DOS, a strictly real-mode operating system. This interface is one of the most critical functions of a DOS extender—even more so with extenders written for the 80286 that have to cope with the difficulties of getting that chip back into real mode, as discussed in chapter 2. When multitasking is involved, extenders provide a common interface that can be shared. There's a lot that must be done.

When DOS services are needed, the DOS extender in some cases will handle DOS calls itself. For others (file I/O, for instance), it generally will switch the processor back to 8086 real mode and let DOS do the work. Mechanisms vary according to the task at hand; however, a look at the way extenders handles a DOS call gives some insight into what goes on invisibly behind the scenes.

A protected mode program cannot be allowed to access DOS directly. To get around this problem, an INT 21h call made from protected mode typically is handled something like this:

- Intercept the INT 21h software interrupt call. (The DOS function interrupt, INT 21h, is actually a collection of standard functions available to all programs that need them.)
- Move any extended memory data buffer into conventional memory.
- Switch the processor to real mode.
- Reissue the interrupt call to DOS.
- Switch the processor back into protected mode.
- Move any returned data buffer to extended memory, if necessary.

Although specific features vary between the various extenders on the market, most can call real mode routines up from protected mode and vice versa. Some can write directly to the screen to save the time that otherwise would be wasted going through the BIOS. Support for virtual memory demand-paging varies from good to nonexistent.

So far, one of the main reasons to use a DOS extender—the main reason probably—has been to allow an application to easily access large amounts of memory. Another and sometimes overriding consideration, however, is the ability to run programs developed under high performance compilers such as MetaWare's High C or other comparable development tools. High performance compilers means high performance programs.

High performance on a grand scale is one of several forces that have spurred the rush of increased interest in and utilization of extended memory and that would seem to give DOS an even stronger position in the long term scheme of things. The slow and seemingly increasingly uncertain emergence of OS/2 has certainly frustrated many ambitions. It's a lot more than that, for with the extended memory tools available to today's software developer, the full 32-bit processing capabilities of the 80386 can be tapped right from DOS.

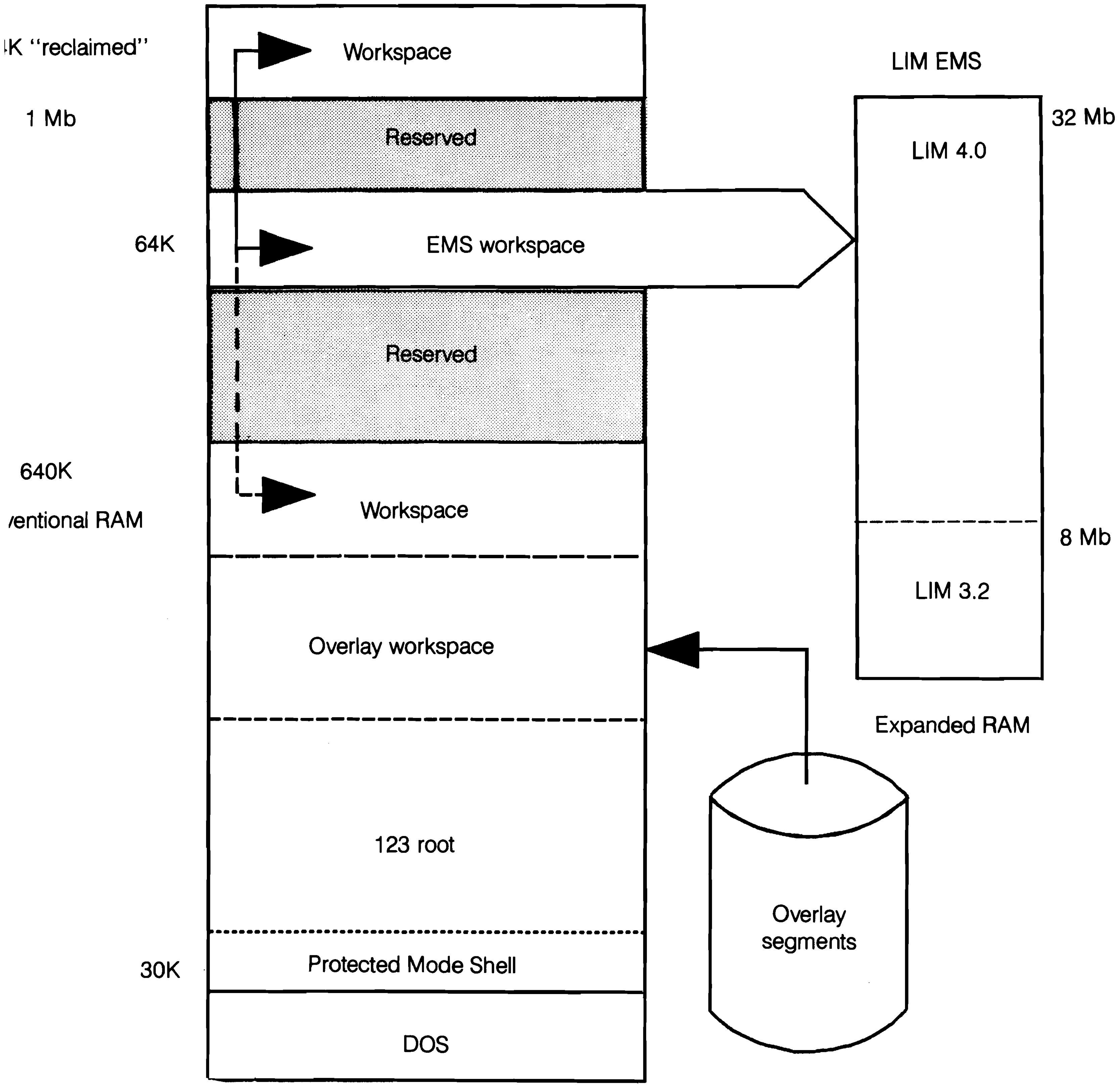
What is emerging then is a new genre of software (in many cases, new releases of old favorites) tailored to run in extended memory under DOS with no exotic and iffy new operating system required. In some cases, two parallel sets of new device-dependent software are emerging, making no apologies for the fact that, although they do run under DOS, they might or might not run on an 8088. Or, in the case of 80386 DOS/extended memory software, on an 80286.

There are several unique advantages offered by going the DOS extender route. Aside from the fact that software developers and users don't have to wait for some new operating system to be developed, the mere fact that it is being done under DOS means that any ordinary software written for the DOS environment can run alongside it, with no gimmicks (like the so-called "compatibility box" in OS/2) and no waiting.

Lotus put it all together—just like 1-2-3

Lotus was among the first DOS applications to offer a DOS extended version for 286s and up. Its version 3.0, developed around Rational Systems' 16-bit extender technology, will run even crammed into 1 Mb, as shown in Fig. 7-2. No longer near the so-called "leading edge" of the technology, a position it held when first introduced, 16-bit extender technology still is an almost ideal example because of the rare insight it gives into the way that the same basic product can run so differently in two different operating environments. It is doubly interesting because Lotus hedged their bets in version 3.0, giving us a product that would run under either DOS or OS/2. (There actually are two different Lotus 3's, one for DOS and one that only has the same look and feel designed to operate under OS/2.)

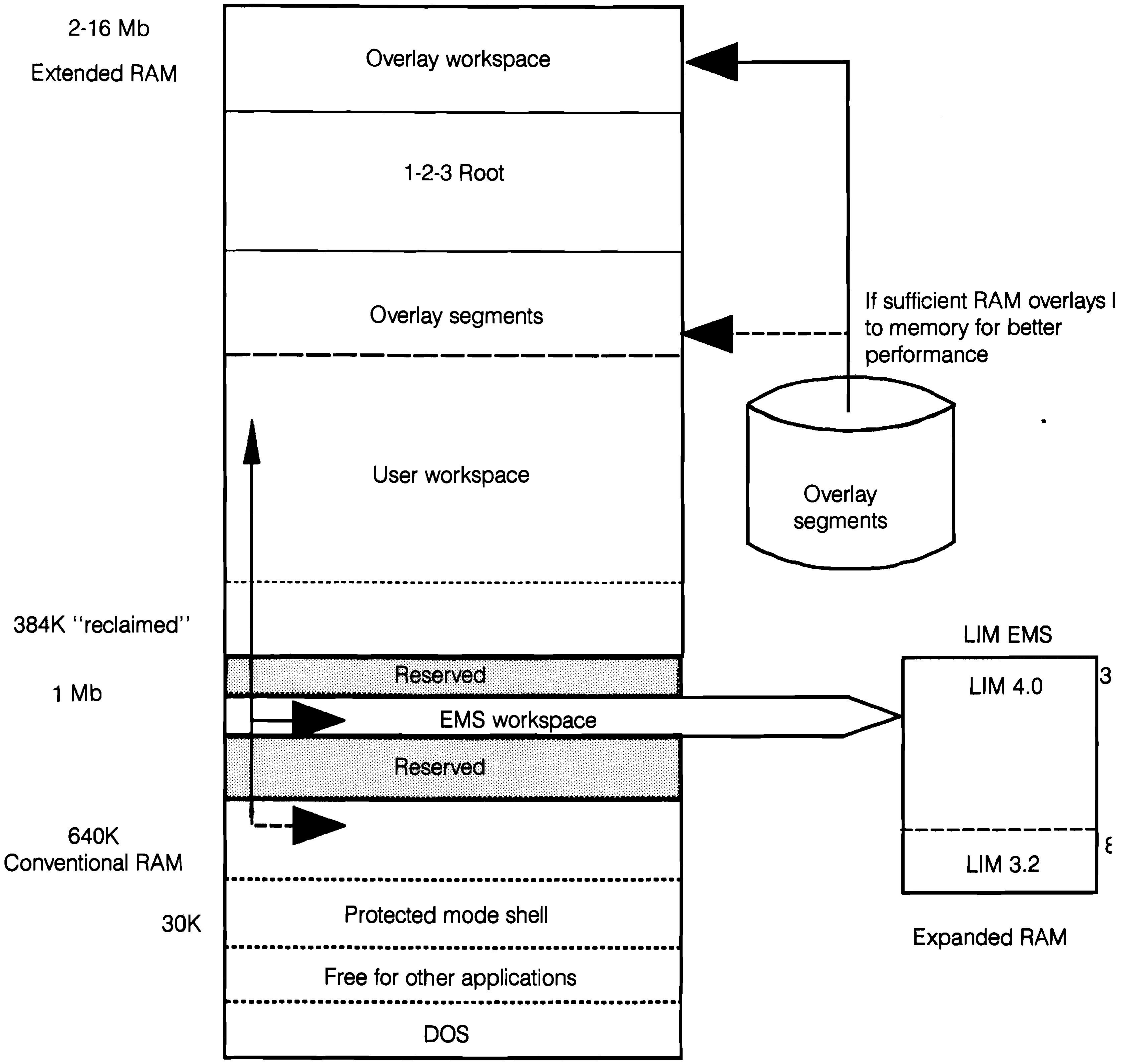
In Fig. 7-2, however, you can see graphically just what extended memory means and what it can do that expanded memory cannot. Note that the 1 Mb is not simply 1 Mb of address space in 8088 fashion, but 1 Mb of installed memory as is typical of many 286 and higher systems. That upper 384K is actually extended


K "reclaimed"
1 Mb
64K
640K
entional RAM
30K
Workspace
Reserved
EMS workspace
Reserved
Workspace
Overlay workspace
123 root
Protected Mode Shell
DOS
LIM EMS
32 Mb
LIM 4.0
8 Mb
LIM 3.2
Expanded RAM
Overlay
segments

DOS-extended Lotus 3 demonstrates how it can fit on an 80286 (or higher) system with only 1 MB plus expanded memory. However, overlays must be called in from the disk as needed, significantly slowing many operations. Memory shown as "reclaimed" is simply the balance of a full megabyte installed on many computers, 384K of which is above 1024K.

memory used here as workspace in conjunction with whatever is available in the lower 640K. Expanded memory can be used, but mainly for data storage.

Lotus 3.0, typically, is too large to fit in one program, at least as far as DOS is concerned, so it's broken into overlays that must be loaded in from disk when they are needed and loaded over when some other task must be performed. This process is cumbersome, but it is the way users are used to doing business with big programs in constricted workspace. Note how the whole game plan changes in the extended memory environment depicted in Fig. 7-3.


2-16 Mb
Extended RAM
Overlay workspace
1-2-3 Root
Overlay segments
If sufficient RAM overlays l
to memory for better
performance
User workspace
Overlay
segments
384K "reclaimed"
LIM EMS
1 Mb
Reserved
LIM 4.0
3
EMS workspace
Reserved
640K
Conventional RAM
LIM 3.2
30K
Protected mode shell
Expanded RAM
Free for other applications
DOS

7-3 Lotus 3 flexes its muscles in an Extended Memory environment. Given sufficient Extended Memory, the overlay loaded into RAM, eliminating the need for repeated disk access. Expanded Memory, however, still is used as space.

Here, the amount of extended memory available (not just the fact that you have some) clearly becomes a determining factor. If you have enough extended memory, the 1-2-3 root, or kernel, is no longer in conventional memory. The root and whatever overlays available memory will allow are all loaded in extended memory. The program no longer has to run back to the disk for another function overlay all the time. It's all there in memory and available at RAM speed. There's workspace—lots of workspace up there, provided you've got RAM enough.

For data storage, there still is up to 32 Mb of LIM 4.0 expanded memory to fall back on. In the case of Lotus 3.0, it is the full 32 Mb, not the archaic 3.1-type expanded memory access Lotus supported through version 2.1. All this memory yields a grand total of up to 48 Mb on an 80286. Expanded memory is being used but has been relegated to a supporting role, because only extended memory can provide the environment needed (no bank switching or waiting for overlays). In this scheme, expanded memory could be the bottleneck where large spreadsheets are involved.

I cited Lotus here not because it is unique but more because it is not. Eliminating the bottleneck caused by expanded memory is the lure of extended memory. This is why more and more developers are looking to it, banking on it for the future.

Lotus's entry into the DOS-extended arena also focused attention on some other issues that developers must face (the one I discussed earlier in this chapter in particular: the need for some common, industry-accepted interface). The DOS extender Lotus used conformed to the VCPI, virtually assuring its compatibility with DESQview and its ability to multitask in extended memory along side more mundane DOS applications.

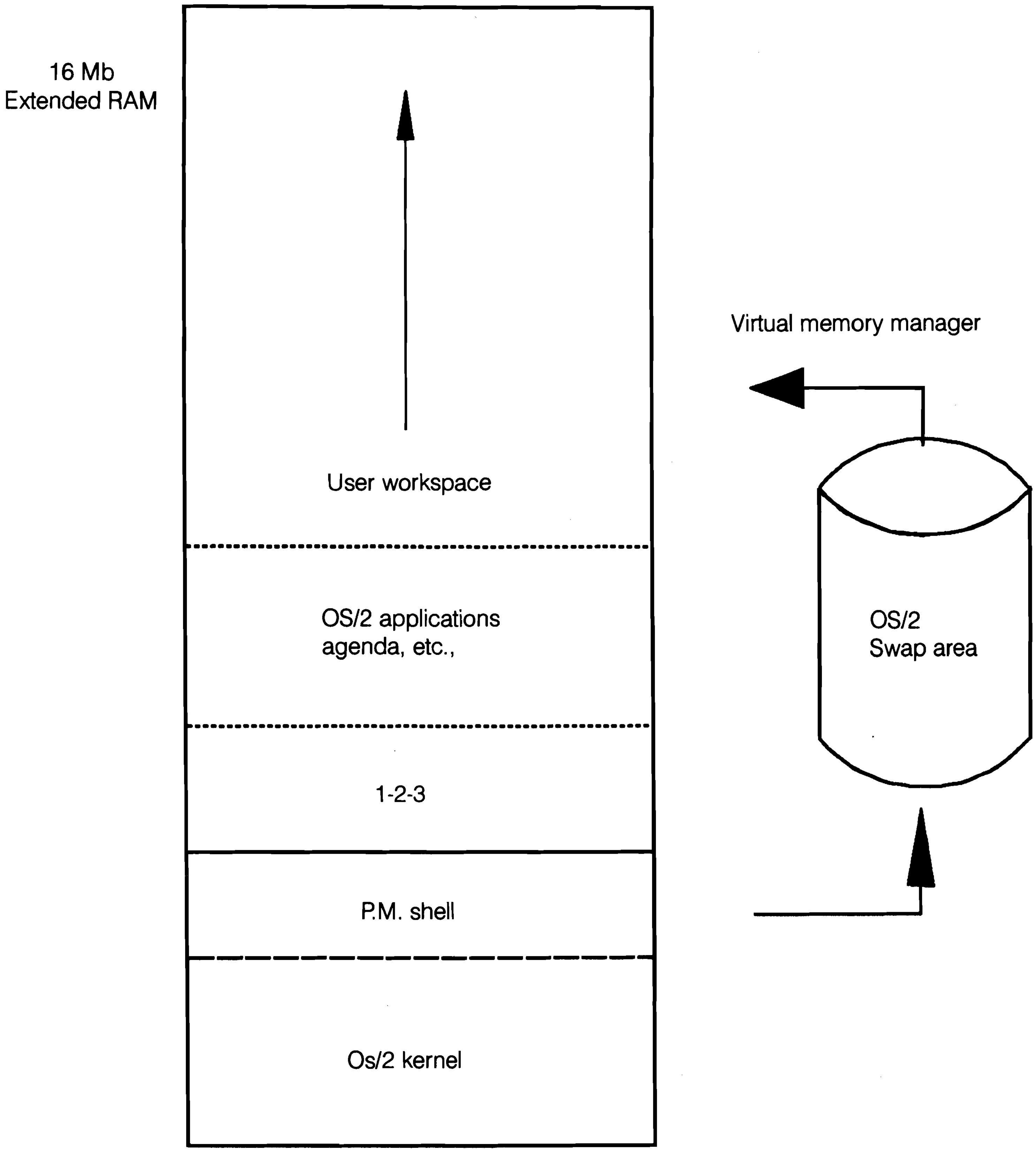
This compatibility requires some careful interfacing because the 80286 must constantly be switched back and forth between real and protected mode operations. But Microsoft's Windows did not support VCPI, the then-industry-accepted interface standard, at the time Lotus announced version 3.0 and refused comment. After sitting on the sidelines watching the rest of the world seemingly pass it by, however, Microsoft dropped the other shoe in the spring of 1990 with the introduction of another, rather different interface specification, which I will look at later in this chapter, the DPMI.

Before leaving Lotus, I should turn your attention quickly to one other feature of version 3.0: OS/2 compatibility. Lotus doesn't have to run under DOS. Although DOS is the subject of this book, a brief aside seems called for. As you'll note in Fig. 7-4, under OS/2, Lotus will run in the linear memory called extended memory, using a virtual memory manager instead of expanded memory as a swap area. By any name, extended memory is where the action is.

Canned answers for uncanny problems

One of the main reasons for the rapid emergence of DOS-extended software has been the ready availability of the means of accessing extended memory from DOS. Initially, some developers—Oracle, for instance—went their own way, writing their own proprietary DOS extenders. Developers, however, can buy DOS extenders basically right off the shelf. Libraries of custom modules along with special linkers and debuggers make the venture to the outer limits relatively painless, whether developing new products or enhancing an existing product.

Many programmers claim that using libraries is much simpler and faster than trying to get a product up and running under OS/2. Furthermore, programs and


16 Mb
Extended RAM
User workspace
OS/2 applications
agenda, etc.,
1-2-3
P.M. shell
Os/2 kernel
Virtual memory manager
OS/2
Swap area

7-4 A hint of things to come, Lotus 3 also will run under OS/2. In that environment, Lotus 3 uses linear memory as workspace (up to a total of 16K, the limits of an 80286) for even faster operation. Lotus then uses the disk as a virtual memory swap area when RAM is not sufficient for its needs.

programming languages that had never migrated to DOS because of segment and size limitations now can be ported to DOS. (Catering to the architecture of the 8088 and 80286, DOS assumes that memory is broken into 64K segments, no matter how large that memory might be.) In many cases, eliminating EMS management code and recompiling is all that's required to gain the added speed and performance of running in extended memory using the full 16 or 32-bit processing capabilities of the system—typically two to three times faster on a 32-bit system than an 80286, and much faster there than on an 8086/8088. The canned answer would seem to have a whole lot going for it.

There are three names that keep coming up in any discussion about off-the-shelf DOS extender developement tools: Ergo Computers (formerly AI Architects), Rational Systems, and Phar Lap Software, Inc. Of the off-the-shelf 32-bit runtime environments currently available, AI Architects' OS/386 seems to offer the closest emulation of DOS and BIOS. Most DOS calls are fully supported, except that 32-bit registers are used. Each environment, however, has its merits and its supporters.

Borland went to a Phar Lap when it decided to release a special 386 version of Paradox—a release that runs up to five times faster than its 16-bit version.

Lotus, after putting its name to the LIM EMS, which its product did not support past 3.1 EMS, jumped on the DOS extender bandwagon with its version 3.0, as cited earlier. While at the same time offering a superficially improved 2.x for its established PC user base, Lotus still took a more cautious approach in version 3.0, incorporating a Rational Systems DOS extender that would not exclude the 80286 market. That particular extender includes segmented virtual memory support that allows for swapping certain data temporarily to disk if there is insufficient RAM available or if the need exceeds the 16 Mb of RAM memory addressable by the 286.

IBM now is also marketing software developed around Phar Lap's extender. This one was especially interesting from several standpoints. What makes its Interleaf Publisher desktop publishing software especially interesting here is that it was developed not under OS/2 as anticipated by industry pundits, but rather under DOS. This choice was hardly a vote of confidence for the future of OS/2 and, according to some, probably another nail in the coffin.

Phar Lap concentrated mainly on the 386 arena. Other software companies, most notably AI Architects and Rational Systems, offer DOS extender development tools for both 16- and 32-bit runtime environments. Most interest, however, seems centered on the 32-bit environment.

With CAD, CAE, and a host of scientific and technical applications, the list of DOS extender users today is almost endless. There are literally hundreds of them. This amount is certainly enough to demonstrate—if there was any doubt—that DOS extenders are serious software tools for serious work.

Programs executable in extended memory via some of the better known DOS extenders characteristically have file extensions different from the .EXE and .COM tags we're used to under DOS alone. AI Architect's protected mode programs use the file extension .EXP (EXecutable Protected mode). Phar Lap unfortunately chose the same extension for 32-bit mode files created around their extender, which would be fine except for one thing: Phar Lap files are structured differently. AI Architect's software, however, can run Phar Lap's .EXPs by calling the .PLX (Phar Lap eXtended).

Regardless of what's going on inside, as far as the user is concerned, everything is done from DOS just like it's always been done. All the other things work just the way you think they should, even if you're using a control program like DESQview from Quarterdeck Office Systems, now that we've got the VCPI.

What's a VCPI?

Initially, even having the techniques at hand to write workable extenders was only half an answer. The problem ultimately wasn't even so much between the various DOS extenders that emerged so much as between applications using DOS extenders to run in protected mode and control programs, such as DESQview. Some of the problems included microprocessor switching, hardware interrupt processing, and the sharing of extended memory.

The acronym comes from Virtual Control Program Interface. (The "virtual" referring not to the kind of program but rather a control program for virtual machines, that intriguing third mode of operation.) The conflict specifically was and still is between programs running in protected mode alongside programs using the virtual 8086 mode of the 386. For example, DESQview 386[23] creates a separate virtual 8086 machine with its own megabyte (plus expanded memory) for every application loaded into it—or at least for every window (some windows might have more than one application loaded into them, as in the case of TSRs loaded ahead of some primary application). DESQview uses an EMS emulator to control memory beyond the normal DOS 640K limits to support virtual 8086-mode operations.

Without some sort of interface that resolves these issues satisfactorily, a control program nust be turned off for the user to run a protected mode application. It's that simple—or complicated, depending on your point of view.

Borland's Paradox 386, wrapped around Phar Lap's 386/DOS extender, originally could not run under DESQview. Quarterdeck (developer of DESQview), however, was one of the prime movers in an effort to bring the two conflicting environments under some mutually acceptable set of rules. Aside from Quarterdeck, other initial sponsors of what came to be known as the VCPI were Phar Lap Software, Inc.; AI Architects, Inc., Quadram, Inc.; Qualitas, Inc.; and Rational Systems, Inc. (Aren't you glad they didn't try to make an acronym, like LIM, from that one: QOSPLASIAIQIAIQIRSI!)

Although it primarily addressed conflicts between expanded memory managers and programs running in extended memory, the VCPI was a major step. For the first time, there was at least a set of reasoned guidelines founded on the combined experience of the companies that had a hand in drafting it. What it said was that any program adhering to the VCPI standards could coexist with any programs running—multitasking on the same machine, but in virtual 8086 mode. In the happy endings department, with a standard and a few adjustments here and there to make accommodation for each other's needs, Paradox 386, with the VCPI, ran nicely under DESQview.

Having a standard and having everybody accept it as a standard are two different things. For reasons that are not entirely clear, Microsoft chose not to accept the standard. Having chosen not even to be party to the drafting of the VCPI specification, Microsoft found its perennial bridesmaid, Windows, left out in the cold and incompatible with any software that supported the VCPI specification. The problems were not unforseen, but extended memory is serious business. Much too serious for Windows to block the view for long.

DPMI: a light in the window

A window, however, has two sides and which side is in and which is out depends on the side you are standing on. Whichever side is which, Microsoft clearly did have several problems with VCPI because there was more at stake than Windows. There were other issues raised by the VCPI of concern to others, as well.

WIth pre-release work on an all-new Windows already at an advanced state, Microsoft summoned other industry leaders and persuaded them to join in a Microsoft-sponsored interface specification, the DPMI (DOS Protected Mode Interface). Obviously, Windows compatibility was of paramount importance to Microsoft.

High on the agenda, however, was yet another important feature of the 80386 architecture that was overlooked by the VCPI: the 80386 provides for multiple levels or rings of protection that can be managed on a need-to-know basis analogous access to the warroom at the Pentagon. Only those with specific clearance can get past the first level of security. Increasingly higher security clearances are required as you get closer to the center, with only a privileged few having full access.

In the 80386, this inner sanctum is called *ring 0*. If you can get to ring 0, you have full access to everything the chip possesses. This highest level is the operating system level. Obviously, the operating system, whether DOS or UNIX or whatever, has to have free access to ring 0 because any less would limit its effectiveness by leaving certain features off limits.

Under the VCPI, however, access to this inner circle was not limited exclusively to the operating system. To a large degree, the VCPI bypasses the protection levels of the outer rings, allowing access to users who might not be properly qualified. A poorly written program, or one that just encounters a problem, with access to ring 0 can bring down the whole house of cards, which is a polite way of saying crash the system.

Now, consider all of the virtual machines you can create using the virtual 8086 mode of the chip. You can create multiple machines for multitasking and for multiuser systems (where a system crash could be extremely costly and could take down the design department, bookkeeping, inventory, and the steno pool in one fell swoop.

Microsoft and others argued that only the operating system—in multitasking and multitasking environments, only the host operating machine—should be allowed ring 0 access. Virtual machines and applications should have no more access than is absolutely necessary. By properly limiting access, a problem application could not crash the entire system, but at worst, bring down the virtual machine that it was running on. If someone in the warehouse screws up, then the computer for the inventory might go down, but everybody else keeps right on working as if nothing happened. As far as they're concerned, nothing has happened.

Although, in practice, the number of problems are relatively few, they do occur. When they do they sometimes, but not always, bring down the system with the loss of any unsaved data. The issue is not insignificant, assuming a much greater importance as you move beyond simply multitasking to multiuser systems of increasing complexity.

There are other issues, as well. The DPMI specification is far reaching, even including features that can bridge the gap between DOS, UNIX, and OS/2 to a point where applications in the future might not have to be classified on the basis of operating system but simply on the basis of the hardware they require. That is for the future. For now, there are some practical realities to face.

In these muddy waters

At this point, there is a double standard among those who were most active in the writing of the VCPI specification still supporting the VCPI. Certainly the differences between the two standards are of such a nature that those who have embraced the VCPI scheme cannot just slip a different module in their code and switch to DPMI support instead.

Interestingly, however, although these two schools of thoughts are not compatible, they are not mutually exclusive. At least two of the most powerful memory managers, QEMM from Quarterdeck and 386MAX from Qualitas, are written to the VCPI specification. Both, however, fully support Windows 3.0, which supports the DPMI instead. Not only do they support Windows 3.0 but also significantly enhance its memory utilization.

On closer inspection, however, this seeming duality is not necessarily inconsistent. Looking to the protection ring 0 issue that under DPMI is reserved only for the operating system but addressable directly by VCPI software, memory management certainly would seem to be on par with DOS for having a justifiable right to full direct access privileges when dealing with the CPU. A spokesman for Qualitas stated rather flatly that Qualitas has no plans to change. Windows 3.0, on the other hand, runs on top of that environment. Allowing the lesser privileges allowed under the DPMI to Windows and to applications running under it is not illogical.

To put the matter in perspective, what would seem more important than the mere facts of who supports which standard is to what use the supporters would put the standard to. It is an interesting issue and one that does not seem likely just to fade away.

An issue that will fade quickly never really was an issue. Even from the beginning, most memory expansion boards could give a choice of either extended or expanded memory, or a mix of both. Extended memory was available long before many of the users had any way of using it or even any reason to understand the difference.

There is even better news when buying new memory expansion boards now that users specifically want and need extended memory. It's a lot easier and cheaper to build a good board for extended memory than for expanded memory. A spokesman for a firm that prides itself on having developed and marketed some of the finest, fastest, most sophisticated expanded memory boards available, summed it up quite nicely when he said, "It would be pretty hard to screw up [an extended memory board]." All those registers that are an issue with expanded memory just don't exist.

I think you can get some idea of the difference in complexity not only of the issues but also of the drivers required by just comparing the sizes of the new MS-DOS 5.0 drivers: 11,120 bytes for HIMEM.SYS (extended memory) and 91,210 bytes for EMM386.COM (expanded memory). Admitted, this point is a crude comparison and there are issues that might be masked.

Getting the extended memory that EMM386 has to have to start with clearly is the easy part. Once the starting address for a block of chips has been established during the configuration or reconfiguration of your system, the addresses don't change. All the software manager has to do is provide an interface.

8

CHAPTER

DOS's mysterious "extra" 64K

For a long time, everybody thought DOS was strictly a 1 Mb operating system. It is but it isn't. There is an extra 64K just above the 1 Mb limit that DOS can use on 80286 and higher systems. Beginning with version 5.0, DOS can even load most of itself up there.

Called the High Memory Area (HMA), this unique memory resource can be used in combination with ordinary extended or expanded memory with both or with neither one. Extending up as high as 1088K (11FFFh), the HMA can be used in combination with EMS memory mapped to unused DOS address space between 640K and 1024K. "Extending" is the operative word because it is closely related to—actually created from—extended memory, but yet it is quite different.

True, DOS only can deal with 1 Mb of address space, which is why this High Memory Area really is a separate issue and must be dealt with separately. It's there though. With a little sleight of hand, DOS can get its hook into an extra 64K block of usable DOS memory by using legitimate addresses right at its outer limit. This assumes that:

- There is some extra RAM available to use.
- The CPU has enough address pins to deal with more than 1024K of discreet addresses.

That immediately rules out the 8088s because, with only 20 address pins, they cannot handle anything beyond 1024K, period. For the same reason, 8088s will never be able to use extended memory, which is the memory area from which the

RAM for this extra 64K segment must be drawn. Even one more address pin (and a bunch of other internal goodies an 8088 doesn't have) would allow access to this high memory area. One more pin would exactly double the address range. The first chip that meets this criteria is the 80286, with not one but four more address pins.

There's a lot more to it than just having CPUs that have the capability of reaching higher addresses, however, because DOS really is a 1 Mb system. The 1 Mb limitation of DOS is why it was a couple of years after the 80286 and limited access to extended memory was available before anybody even found it and, having found it, figured out a way to use it.

With a little sleight of hand

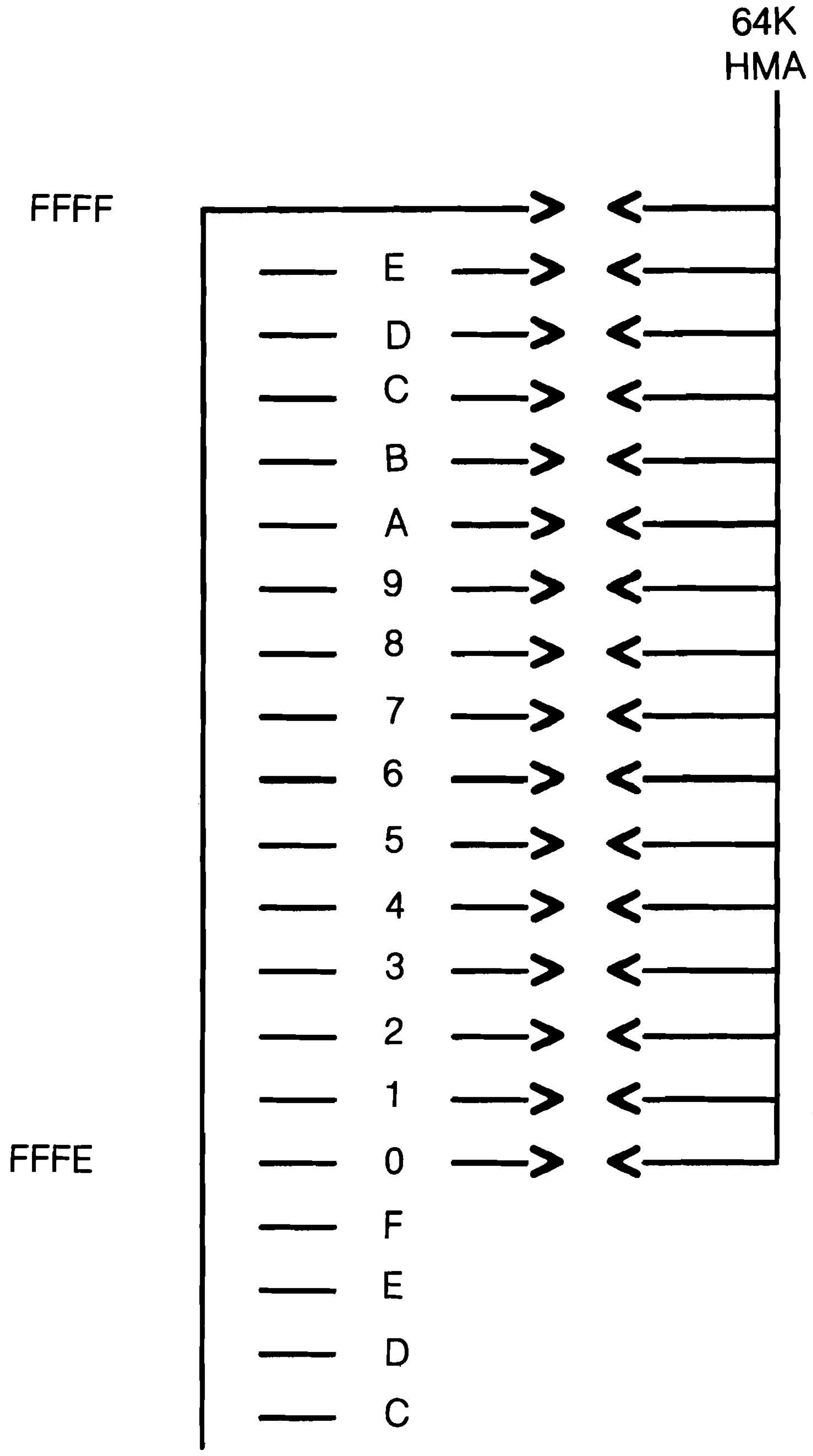
If you will recall from the previous chapter, there are two ways to access extended memory: from the bottom (VDISK fashion) or from the top. At the risk of being repetitious, there is a point at which one byte is the top of DOS (FFFFh) and the next byte, consecutive to it, is the bottom of extended memory (10000h).

In OS/2 or other operating systems compatible with the 8086 clan (except 1 Mb DOS look-alikes), this point would be no more noteworthy than when the odometer of your new car rolls over from 99.9 to record its first 100 miles. To DOS, however, this is a big deal because in going from FFFFh to 10000h, another digit has been added. DOS wasn't written to accommodate another digit. There are ways to work around the problem obviously, but it takes some doing.

I must go back now to something I glossed over earlier: the fact that DOS works in terms of 64K segments. Normally, we tend to think of these 64K segments as starting at addresses that are multiples of 64K. (It's easier to think of segments starting at nice, neat addresses like 0000h, 1000h, and so on.) In truth, they don't have to. Like most rules, DOS is full of loopholes. These loopholes really are what keeps DOS going—not that it's really all that good, rather that there are just so many loopholes. All the 1 Mb limit really means is that you can't have a 64K code segment that doesn't start inside of 1 Mb.

Just for the sake of starting an argument, suppose a 64K segment starts at FFFEh, only 16 bytes short of DOS's 1 Mb limit (Fig. 8-1). Would DOS be able to deal with it? The answer, interestingly is yes. Just as long as the starting address of the segment is a legitimate DOS address.

As most users know, the top 64K of DOS (F000h to FFFFh) is set aside for system ROM and is off limits—taboo. The system ROM does not use the entire 64K, rather only bits and pieces of it—most of it, but by no means all. As long as ROM or something else that cannot be written over without dire consequences isn't using it, the 64K is fair game. (Quarterdeck's QEMM memory manager searches out and utilizes any 4K block of unused address space lower in the ROM area to load part of DESQview's code or to load a TSR or device driver.) Pro-



8-1 Rules are made to be broken. The HMA breaks the rules by starting a new 64K memory segment just 16 bytes below the top of DOS's nominal megabyte. Even though calls to addresses above FFFFh must pass through the A20 gate, which normally separates real mode from protected mode, DOS can use this segment in real mode. The key is that the segment must start within DOS's 1 Mb limit (i.e., below FFFFh).

grams that use the HMA don't care what, if anything, is in the tiny part of ROM overlapped by the HMA segment, only that the offset is large enough to leapfrog past the ROM. To loader access any of the data in a segment lying even partially beyond the 1 Mb barrier working from DOS requires not only a CPU chip that can read from and/or write to addresses beyond 1 Mb but also a set of special instructions—a superset that is compatible with DOS but that also goes beyond the 1 Mb limit. The key, though, is that you've got to start this extra segment from within 1 Mb.

That is exactly what Quarterdeck did as far back as 1986 when it was anxiously looking for someplace—anyplace—to load and run DESQview without stealing gobs of precious memory from user applications. Quarterdeck reasoned that, if a CPU chip had more than 20 address pins (1024K) and memory with contiguous addresses beyond 1024K, it should be possible to start a 64K code segment near the top of DOS's 1 Mb limit with just enough inside for DOS to get a toehold but with the rest of it—all but about 16 bytes—beyond the 1 Mb "limit." Quarterdeck was right.

Needless to say, Quarterdeck jumped at the opportunity. By April of the following year (1987), it was shipping a special proprietary device driver for running DESQview on 286 machines. The driver that performed this "magic" was called QEXT.SYS.

Still, actually using the 64K it does take some doing though because, the 80286 CPU chips (and higher) have something called an A20 gate to screen out dummy calls beyond 1 Mb and to wrap them back around to zero—as was commonly done by many programmers at one time. Every legitimate call to an address above 1 Mb first has to open the A20, then close the door on the way out—quietly. No crashes please.

Although really quite simple, even in its original form, the QEXT driver was highly effective. It made itself look like a 64K VDISK, so the memory area was reserved and other programs wouldn't use it. QEXT also kept an eye on other extended memory functions, so if a program (like VDISK, which can coexist with QEXT) did some work with extended memory, then the A20 line would not get turned off. QEXT would be active doing these things only while DESQview was running.

This process is done without leaving real mode. With the A20 gate enabled, it's really mainly a matter of being able to generate addresses beyond 1 Mb—addresses that require more bits than ordinary DOS software is geared to.

For a time, Quarterdeck, although it made no secret of what it had done, was the sole owner of this high memory area, but not for long. A freebie like this was just too good for a lot of people to pass up. Microsoft put its oar in the water, announcing that it had found an extra 64K of memory to work with—the same 64K—when Windows 2 came out in 1988. After that, the existence of the HMA was codified in the Extended Memory Specification (XMS) released by Microsoft. Along with this specification, Microsoft released a rudimentary device driver very similar to DESQview's QEXT.SYS. Microsoft calls its driver HIMEM.SYS, which the company describes as an Extended Memory Manager (XMM).

This memory could not and cannot be used in any other way. It would not go to waste (unless we let it) because if the HMA is not excluded from extended memory, it is used as extended memory. What sets this block apart as a unique memory resource is that you can use it for DOS and without leaving DOS because, properly accessed, it belongs to DOS.

Even as first released, QEXT.SYS could load a big chunk of DESQview's code beyond 1024K. It set up a 64K code segment starting just 16 bytes inside DOS's 1 Mb outer limit. That gave DESQview almost a full 64K of extended memory, for a total just 16 bytes shy of 1088K. With further refinement, Quarterdeck now can load over 63K of actual DESQview code up there and run with it. To date, no one has done it better or used the HMA more effectively than Quarterdeck does with DESQview.

Fool's gold

As a practical matter, Microsoft added nothing that was not already known and in use when it released the XMS specification. DOS did not formally embrace the HMA prior to the release of version 5.0. Microsoft's acknowledgment, however, formalized the legitimacy of the area between 1024K and 1088K as a unique memory resource. With that, a number of other software developers have released products that can reach beyond DOS's nominal limits to embrace a few more precious kilobytes of memory accessible directly from DOS. The race was on, but all that glitters isn't . . . .

Although the high memory area is closely akin to IBM's VDISK in some respects, it also is quite different. It is different in one critical way in particular: VDISK, like any RAM disk for other virtual disk, can hold any number of different programs or files simultaneously at any given time (up to the total of the RAM available). You can keep cramming them in until you get DOS disk full error message.

You cannot store multiple files simultaneously with the High Memory Area however. The HMA can hold one program only—just as VDISK is one program. If you want a file you've stored on any virtual disk, you do not access that file directly, rather you access the program that creates the illusion of being a disk and access your files through it. Most users never really stop to think about it because DOS handles all the details invisibly for them, but that's what is happening.

You can load only one program into the HMA because only one can have a legitimate DOS starting address below 1024K (FFFFh). You could conceivably load some stupid little 1K or smaller TSR up there, but that would totally waste the other 63K, with no way to recover it in any way DOS could benefit from directly.

Ideally, you would like to find some program that used exactly 64K and not one byte more or less, but that is highly unlikely. If you are going to use the high memory area most effectively, it is important that you load the biggest under-64K program (or divisable portion of a program) to waste as little of the block as possible.

This point is where you come in. As more and more software developers set greedy eyes on the HMA, several seem committed to the credo that their use of it is by some divine right more important than anyone else's could be, no matter how much or how little of the HMA their software actually uses.

As more and more software comes along that wants the HMA, the time is almost surely coming—if it isn't here already—when you, the user, are going to have to make some choices. Even some of the slickest optimizing programs (like QEMM's Optimize or 386MAX's Maximize), despite the job that they can do relocating TSRs, drivers, etc. above 640K, can't help you here. You're on your own and it can be a pretty tricky world out there.

Gift or Trojan horse?

Along with the ability to load most of the DOS 5.0 kernel (47K) into the High Memory Area, Microsoft has cleverly given us a new XMS memory manager, HIMEM.SYS, with DOS 5.0 (similar in name but quite different for the XMS driver supplied Windows 3.0). If you really hope to get the most out of your system, this manager is one gift horse you had best look at really carefully before you let it in your system.

Admittedly, with HIMEM.SYS (or almost any good third-party XMS driver) installed and a DOS=HIGH statement in your CONFIG.SYS, DOS 5.0 now can load all but about 20K of its overhead into the High Memory Area. At first glance, this arrangement is a tremendous boost, as shown below. With DOS 5.0 installed in the conventional way (with the kernel loaded into conventional memory), MEM shows:

```
 655360 bytes total conventional memory
 655360 bytes available to MS-DOS
 590864 largest executable program size

3407872 bytes total contiguous extended memory

3407872 bytes available contiguous extended memory
```

As might be expected, DOS has used 64,496, leaving you just under 600K for use by applications, etc. By loading DOS into the HMA using DOS's HIMEM.SYS with the DOS=HIGH option, you have all but 17,056 bytes of your total 640K available for use by your applications, TSRs, etc., as shown below.

```
 655360 bytes total conventional memory
 655360 bytes available to MS-DOS
 638304 largest executable program size

3407872 bytes total contiguous extended memory
      0 bytes available contiguous extended memory
3342336 bytes available XMS memory
        MS-DOS resident in High Memory Area
```

This process has gained you 47,440 bytes of precious conventional memory for use by applications software or whatever else you care to load down in conventional memory.

DESQview, as pointed out a little earlier, can load no less than 64K of its code up in the HMA. That's an extra 16K. Letting DOS 5.0 have the HMA, therefore, winds up costing precious memory down low instead of giving you a boost. (I picked on MS-DOS here mainly because almost everybody likes to pick on DOS.)

Digital's new DR-DOS 5.0, certainly one of the best DOS look-alikes to come along, does pretty much the same thing up there, too. (DR-DOS 5.0 was introduced a year or more before MS-DOS 5.0.) Unlike the MS-DOS HIMEM XMS driver, Digital DOS's HIDOS.SYS is not restricted to using the HMA only for relocating roughly 37K of its kernel about 640K. It can use any mappable block it finds that's big enough (40K minimum). Chalk up another one for Digital on this feature.

Digging for gold the old fashioned way

Ultimately the test of how well any software utilizes the HMA—or any memory resource other than conventional memory—is how much space you have left over to run your applications. Beginning with version 4.01, DOS has provided a handy, easy-to-use utility, MEM.EXE, to tell you how much space you have left and more, as shown in Fig. 8-2, where the amount of conventional memory remaining is spelled right out.

```
Conventional Memory :

  Name                 Size in Decimal       Size in Hex
-------------     ---------------------   -------------
  MSDOS               52080      ( 50.9K)       CB70
  SETVER                400      (  0.4K)        190
  QEMM386              2416      (  2.4K)        970
  LOADHI                208      (  0.2K)         D0
  LOADHI                208      (  0.2K)         D0
  COMMAND              4704      (  4.6K)       1260
  DOSKEY               4128      (  4.0K)       1020
  FREE                   64      (  0.1K)         40
  FREE                  144      (  0.1K)         90
  FREE               590752      (576.9K)      903A0

Total  FREE :        590960      (577.1K)

Total bytes available to programs :
590960   (577.1K)
Largest executable program size :
590576   (576.7K)

   3637248 bytes total EMS memory
   2146304 bytes free EMS memory

   3407872 bytes total contiguous extended memory
         0 bytes available contiguous extended memory
   1966080 bytes available XMS memory
           64Kb High Memory Area available
```

8-2 Beginning with release 4.0, MS-DOS has included a MEM utility, which was considerably enhanced in version 5.0. This screen was obtained using the /c switch that was added in 5.0. DR DOS also has a MEM utility that provides much of the same information but in a different—and in some ways, more understandable—format.

Actually the information has been available from DOS since before that even. CHKDSK reports, as sort of an afterthought, the total conventional RAM on the system and how much of it is free:

```
655328 total bytes memory
554128 bytes free
```

MEM, however, is much faster. When it is used with the optional /PROGRAM or /DEBUG switches, MEM provides a good deal of additional information, though more than the average user really wants or needs to know in most cases.

This information is of little value unless you know how much free memory you had available before you started fiddling with the HMA. If the HMA is enabled, you need to start off by disabling it and checking to see how much usable free memory you have available without it to gauge precisely how well any software you might want to load up there can utilize it.

From that point, using the HMA efficiently is just a matter of trial and error. To do it right, you've got to deal with DOS as raw and unadorned as you can make it. The easiest way is probably from a bootable floppy with nothing in the CONFIG.SYS or AUTOEXEC.BAT except the specific software that you want to test. You cannot test the software from within DESQview or Windows because a windowing environment might mask the actual numbers, as in the example shown below:

```
 203776 bytes total conventional memory
 203776 bytes available to MS-DOS
 123168 largest executable program size
3047424 bytes total EMS memory
 327680 bytes free EMS memory

3407872 bytes total contiguous extended memory
      0 bytes available contiguous extended memory
 212992 bytes available XMS memory
        High Memory Area in use
```

This example is exaggerated, but it demonstrates that the size reported might have little to do with the actual amount of memory available but rather only the way the window is configured.

Once you have determined which one of your programs makes the best use of your HMA, there are several possible strategies that you can use to put the one you want up there. Most programs that can use the HMA give you a yes/no option when you install them. DOS 5.0 will only use the HMA if you put a DOS=HIGH line in your CONFIG.SYS.

MS-DOS 5.0 also provides another mechanism for controlling access to the HMA. If the HMA is not used for the DOS kernel, you can specify some minimum size that will prevent any program smaller than the size you set from loading in the HMA. This size is specified in the CONFIG.SYS on the HIMEM.SYS command line as shown here:

```
DEVICE = HIMEM.SYS /hmamin = nn
```

where *nn* is the size in kilobytes

This method, however, is probably the least effective one because, unless you know specifically what use your software could make of the High Memory Area, you don't know whether you should set /hmamin = to 5K or 50K. You don't even need the /hmamin = if you take more direct means of controlling HMA access.

However you go about it, the name of the game is to try to find whatever combination gives you the most usable conventional memory to run your applications, after DOS and everything else is loaded. In the HMA, the game rules are a little different, but the object is the same: whoever winds up with the most bytes wins. So far, DESQview comes up the hands-down winner when it's in the game—but then Quarterdeck was the company that found the HMA in the first place. Beyond that you're strictly on your own.

9

CHAPTER

Chairmen of the board

How much memory you cram into your system doesn't matter. Whether it's just that extra 384K left over from the megabyte your system probably came with or a flock of megabytes, without a memory manager it isn't even there. You might as well try to run your computer without an operating system. Yet so far, I have looked at memory management only tangentially, concentrating mainly on the several guises memory beyond 640K assumes and its various uses.

Although the hardware as pointed out repeatedly, must meet certain minimum design criteria to allow effective software control, beyond that the software determines how effectively that hardware is used. It is a team effort, but both the players and the rules keep changing.

The changing face of management

You might expect that, because more and more TSRs and device drivers have been written or updated to avail themselves of memory of one sort or another above 640K, the situation up there would only get more crowded and confused. If anything, however, the reverse would seem to be true because more and more TSRs and drivers—including some that earlier relocated into upper memory between 640K and 1024K—are moving on out into the suburbs of expanded memory.

While this expansion has been going on, some of the more powerful memory management packages like QEMM and 386MAX have gotten still more powerful and turned what was an art requiring many patient hours on the part of users into a science. Instead of having to spend hours working out elaborate loading strategies, these two programs supply utilities that completely and painlessly automate the process. Not only are they painless, but they often do a better job.

How much better they work varies, and there might be situations where they really aren't better. I know in my case, that, after many patient trial and error hours, I had managed to load some 209K of TSRs and device drivers above 640K, including the High Memory Area. I thought this amount was pretty good until I used QEMM's Optimize. In just under 2 minutes, the program not only squeezed an extra 4K into upper memory but, in the process, gave me an additional 32K of conventional memory.

On the flip side, however, I actually have space to spare these days in upper memory though. In part, that is the result of new releases of some old favorite TRSs and device drivers that require less high memory. In other cases, all-new programs—again generally with smaller upper memory appetites—have displaced old favorites.

Also, to avoid the second-generation RAM cram that was developing in what Microsoft still refers to as reserved memory above 640K, a number of programs have moved sizable chunks of code out into expanded memory, leaving just enough in upper (or conventional) memory to mind the store. There are programs, such as Headroom and PopDrop, that ease the crunch still more, if need be, with swap-out strategies that can be used to stretch whatever upper memory you've got still farther.

The net result is that, at this point, most users could probably live quite happily with just the upper memory mapping capabilities of DOS 5.0's EMS emulator, EMM386.EXE, which, alongside any one of several third-party managers, is something of a wimp. I'm not saying you should rush out and buy MS-DOS 5.0—or DR DOS 5.0, which has similar overall capabilities—and think you've got all of the memory management you want or need. Even if you add a network card, a data compression board, and maybe a special video adapter or mass storage media device, things still can get overcrowded up there. You also could change your software, start using Windows 3.0, and see how little space it wants to leave for any other programs up there. For these and other reasons, I will look in greater detail in later chapters at the extra horsepower offered by some of the top third-party memory managers, that can be well worth the added cost.

Duos, quartets, and one-man bands

In prior chapters, you have seen four different kinds of memory beyond 640K, which means there are four kinds of memory to manage. Granted, it all comes down to two basic kinds, extended and expanded. When you strip those two down just a little farther, however, you have four kinds of memory management required to cover all the bases.

Managing extended memory and its companion HMA is easy. DOS 5.0's HIMEM.SYS is just a little over 11K and that does that about as well as anybody's

XMS driver but just for extended memory and the HMA. As stated earlier, however, that is the easy part of memory management.

Only when you dig into EMS expanded memory and begin mapping LIM 4.0 EMS memory to unused address space above 640K (but still within 1 Mb) do things start getting complicated. And that isn't all that starts to get a little complicated if you let it because there are managers that:

- Only do extended memory
- Only do the HMA
- Do both the HMA and extended memory
- Do only EMS memory but do not support UMB mapping
- Do only EMS memory and do support UMB mapping
- Do extended and expanded memory only
- Are specific to proprietary hardware
- Do all four types on 80386s only
- Are for 286s

The list goes on. I think you see the problem, though. The kicker is that you can only have one memory manager installed for each kind of memory that you need managed. There are one or two notable exceptions. In some instances, if QEMM is installed after certain other drivers, it completely takes over from them. In doing so, they do not break the basic rule, however, because only one survives. DOS 5's HIMEM.SYS does a fine job; however, its companion, EMM386.EXE, is no match for QEMM, 386MAX, or any of the better third-party, 80386-specific managers.

Even if you really like HIMEN.SYS, you'll most likely have to dump it if you want a better manager for EMS expanded memory or mapping. Most expanded memory managers are total packages that do everything. (Fortunately XMS drivers are not the sort of thing you can get emotional over.)

On the other hand, there are some 286 managers (including the proprietary manager that comes with All Computer's ChargeCard) that do not support the High Memory Area. To add HMA support, you must add QEXT or some other driver that only does the HMA but does not conflict with any other features of the proprietary driver specific to the ChargeCard hardware. However, the HMA, as interesting as it is, is really only a peripheral issue. There is a bigger picture.

New directions

In looking at the field of memory management, today more than ever before perhaps, you need to look at it in terms of where you are going—or can at least expect to go—from here. More than in any other way perhaps, the real significance of MS-DOS 5.0 is the way it looks beyond the old 640K horizon.

Perhaps the most significant statement MS-DOS 5.0 makes is that not only has extended memory arrived, but extended memory is the future. Expanded memory and the ability to map memory to address space above 640K is supported by DOS 5.0 (actively supported for 386s only) but really only as an afterthought as far as Microsoft is concerned; you must install HIMEM.SYS, the new DOS 5.0 XMS driver for extended memory and the HMA.

There are some people in the industry already predicting the total demise of EMS memory as we know it. That prediction seems unrealistic given the solidly established software base that is dependant on expanded memory and the many users who, by nature of the machines they are using, have no other avenue of escape from 640K open other than expanded memory. There is no reason the two basic forms should not continue to coexist, because expanded memory in no way detracts from the overall superiority of extended memory.

Beyond the many other indications that the industry is moving above and not just outside of 640K, in release 5.0, DOS for the first time has a mechanism that allows most of the kernel—some 47K of it—to be relocated in the High Memory Area. The mobility makes DOS 5.0 an active player in the HIMEM sweepstakes, not just a passive host.

Relocating a major portion of the DOS 5.0 kernel, however, is a function of the kernel and has nothing to do with DOS 5's HIMEM.SYS XMS manager (unless the /hmanin parameter is used and is set to some higher number than the kernel would require). If the HMA is there, the kernel can relocate to it. It makes no difference even which XMS manager you use—QEMM, 386MAX, or whatever. If the High Memory Area is supported, the kernel can be relocated by simply adding a DOS-HIGH line to the CONFIG.SYS.

Granted, the fact that 5.0 makes extended memory its first priority for 80386 systems (now even offering features usable only on 386 and higher systems) is self-serving on the part of Microsoft since Windows 3.0 really is written for extended memory. In the broader picture, however, the rationale clearly seems to be a tacit acknowledgment of a changing world with changing needs that can only be met by linear memory (including the HMA). It also seems to be an acceptance of the fact that the world—even the DOS world—can no longer travel at the speed of the 8088 PC.

Someone once said, "Don't never look back 'cause sumpthin' might be catchin' up." Depending on what you think might be back there, that might be pretty good advice. It's not in this business. Here you'd better keep an eye out over your shoulder or you'll get left behind. These days, that means that no matter what you're doing with expanded memory and all the many benefits it still offers now (and always will, certainly as long as there are 8088s and no doubt even after that), extended memory is here.

Getting down to brass tacks

As you can see from earlier chapters, extended memory is much easier to manage than expanded memory. MS-DOS's new EMS emulator for 80386 and higher systems (EMM386.EXE) is about nine times the size of HIMEM. SYS, its companion manager for extended memory and the HMA. From the user point of view, managing extended memory is so cut and dried today that it should not even be a factor when deciding on a memory strategy.

The real key to effective overall memory management then is still expanded memory, despite the fact that the prime focus of the industry has now turned to the extended memory arena. As long as you run applications in conventional memory and need to maximize your use of DOS's megabyte, you will need expanded memory. Given that fact and the complexity of the issues involved, you should look closer at these issues and the way you manage your expanded memory.

There are proprietary EMMs that come with every board that's capable of giving you expanded memory. DOS is in there too with EMS support for 386 and higher systems only. Then, there are third-party EMMs, especially for 386s, that claim to be better.

Times have changed, hardware has changed, but more importantly user needs have changed. To a large extent, the changes in user needs were caused by an increasing number of programs that now are written to use EMS memory (if it is available), so you can use the memory they used to occupy down low for other things. It is like having your cake and eating it too. Fortunately, while the number of programs using extended memory was growing, memory management technology also changed dramatically, driven in part by a healthy competition between developers of memory managers.

These technological changes were especially dramatic with hardware that fully supported the LIM 4.0 EMS specification, which was no longer limited to just that single 64K page in high RAM. The now legitimate function of backfilling from 640K down and swapping out everything between, say 256K and 640K (including running code, which still is running happily), makes it a whole new ball game that imposes new demands on expanded memory managers. It should match at least the capabilities of any add-in extended/expanded memory boards, but at the same time be smart enough to recognize a computer that does not allow it and adjust accordingly.

Probably about the only users who cannot reap significant benefits from today's more powerful memory management tools are users still running with old 3.2 spec EMS boards. That problem will no doubt continue to haunt us for some years yet to come, especially in the 8-bit 8088 bus area. Certainly, the problem will remain as long as there are buyers more concerned with price than a proven track record.

Even after ruling out those problems, however, there still are significant differences, particularly in the ways different 386 managers allow users to fill in upper memory gaps and provide space to offload system overhead and TSRs. I discussed this specific area in much greater detail earlier where I dealt only with mapped memory. There are some other management issues that need to be examined yet, however.

Super specialists

Beyond the lingering 3.2 EMS problems I have discussed previously there are some brand new memory management problems coming up with new high-quality equipment. IBM, for instance, created a dilly with its PS/2 386 when they took everything from E000h to FFFFh for system ROM—the top 128K rather than the more modest 64K almost everyone else has been able to live with.

In rare defense of IBM, there was a reason for grabbing all that space. Looking ahead to OS/2, designers incorporated a lot of routines up there that have no use under DOS. That reasoning, however, is of little consolation to DOS users, who are stuck with all that system overhead.

Although the EMS page frame is commonly located at E000h, it's easy enough to move it somewhere else. Moving it, however, doesn't leave much for high memory use. So Qualitas took a hard look at that 128K ROM BIOS and came up with a unique solution in a package appropriately named BlueMAX.

Qualitas has, in its own words, compressed BIOS. (This compression should not be confused with the kind of data compression technology that squeezes extra space out of your hard disk.) What Qualitas has done is simply to eliminate the OS/2-specific portions of the BIOS from its rewritten version. Qualitas also threw out ROM (Cassette) BASIC, that throwback to a time before PC's were even supposed to have floppy drives. Any PS/2 386 users who wish to use BASIC can use GWBASIC, Quick Basic, or most other stand-alone BASIC interpreters or compilers instead of BASIC or BASICA when using BlueMAX.

Even in a normal 64K ROM region, there is a lot of empty space (a few hundred bytes here, a few hundred someplace else adding up to several kilobytes). The total amount of memory saved or gained by all this pruning is in the order of 80K—80K of contiguous mappable address space, or about 65% of the original size of the reserved BIOS region. Calls for any of the routines retained by the compressed version of the BIOS have to be revectored and RAM-mapped to the scavenged address space before they are of any use to you. These are pretty routine management functions.

Combined with its other memory management features, BlueMAX often can make available as much as a total of 212K of contiguous high RAM during the initialization phase of booting a PS/2. (This number includes the 64K in the page frame that can be borrowed for use when installing programs like DOS's FAST-

OPEN that require such huge chunks to initialize but require very little to actually run.) The actual total varies from machine to machine depending on any network or other adapters that might be installed. In any event, it is impressive.

This is doubly interesting because the same kind of BIOS compression employed by Qualitas in developing BlueMAX is applicable to any ROM region. Qualitas started with the PS/2 because:

- Its BIOS region is so large
- All PS/2 386s have essentially identical genuine IBM ROMs

Unfortunately, with the great number of other BIOS ROMs in use by different IBM machines and clones (and even clones of clones), it seems improbable that a BIOS compression scheme can be developed that is sufficiently generic to work with all or even any major part of these systems. Rather, it seems they would have to be taken on a case-by-case basis. At best, the saving starting with a 64K ROM region would be smaller but still significant, especially if—as in the case of the BlueMAX—the revectoring of the compressed BIOS resulted in a larger contiguous block of mappable addresses.

In the highly competitive third-party memory management market, Quarterdeck, not to be outdone, introduced its own PS/2-specific memory manager: QEMM50/60. Not content with going toe-to-toe with Qualitas to stay in game in the lucrative PS/2 market, Quarterdeck set its sights on recovering the whole ROM area.

Gentlemen, start your engines

With the release 6.0 of QEMM 386, the race for the ROM region took on a new dimension. Rather than merely compressing ROM to make it fit a smaller block of precious real estate, Quarterdeck elected simply to remove it. (It was not so simple in terms of the technology involved, but certainly the most innovative approach successfully implemented to date.) Not a simple upgrade from the version 5.*x* QEMM distributed up until about the time MS-DOS 5.0 was released, release 6.0 would seem to represent the vanguard of the next generation of 386/486 DOS memory management technology.

The key to Quarterdeck's new QEMM386 version 6.0 is its new QEMM-386.SYS driver, which can load all but about 4K of its own overhead above 640K. The real story is in Scram!, one of two new powerful support utilities in this release. Scram! simply declares open season on ROM.

Scram!, by moving ROM out of the upper memory region, typically can increase the amount of available mapped memory by 98K to 128K to free that address space for other usage. This applies not only to system ROM (typically F000h to FFFFh, or E000h to FFFFh on PS/2s) but, in many cases, video ROM and hard disk ROM as well, giving you a clean sweep and a block of contiguous memory the likes of which you've never seen above 640K.

For some time, Quarterdeck's memory managers have been able to spot certain parts of the system ROM region (differing from system to system) that could be mapped over. That was simply a case of finding regions that were not needed—or at least not needed after the initial bootup phase—and mapping over them. This scheme is entirely different. This scheme, using any of three different strategies, moves all ROM out of the way.

To put things in perspective, not everybody really needs this total power. As in everything else you do with your computers there is a tradeoff involved. The memory you use for mapping has to come from somewhere. The more fully you utilize these capabilities, the less EMS (or extended) memory will be left for normal expanded or extended memory uses. For this reason, these are options that can be enable or not at the users option.

No matter how you look at it, however, the combination of features available in QEMM with this release represents the most powerful overall DOS memory management package seen to date. However, it also clearly seems to demonstrate the fact that software is fast approaching the ultimate limits of what can possibly be done under DOS.

Scram!'s three strategies generate the same amount of additional mappable memory but differ in their approach. One strategy exploits memory management and protection features of the CPU. This is called the *protected method* (MR:P). Alternately, Scram! can use an EMS equivalent, which is called the *mapping method* (MR:M).

Scram!'s third strategy will generally make less additional memory available than the other two. Depending on the individual system, it might be more compatible, so this is called the *compatibility method* (MR:C). This mode attempts to share the EMS page frame block with a selected ROM area. While QEMM, by default, will do the choosing of what to map where, it still might be necessary to fine-tune your individual configuration with a FRAME= parameter for the best results.

The power to map ROM produces some interesting side effects that might not be immediately apparent to many users. Copying ROM to RAM—often referred to as shadowing—has been an accepted technique for some time, speeding the execution of commands that would otherwise have to be read and re-read from relatively slow ROM. By extending this to video ROM as well, you can significantly improve video performance in many cases as well (something that should be of particular interest to anyone running graphics applications or using a graphics interface).

Unlike various attempts to provide additional contiguous memory for DOS applications by revectoring the video itself—notably by Memory Commander and previously (although it is no longer supported) by ALL computers in conjunction with the ALL ChargeCard—Scram! leaves the actual video region unmolested.

However, the graphics video can be mapped by QEMM as mentioned elsewhere in this book, but that is a separate issue.

In addition to Scram!, Quarterdeck has another utility called Squeeze!. Squeeze! can make certain previously off-limits areas above 640K available while loading and initializing TSRs that run in less space than is required during loading. Although Quarterdeck has taken a somewhat different approach, something similar had been done before and is discussed elsewhere in this book.

Quarterdeck also enhanced its automatic Optimize utility, adding not only support for these new features but several others, including a new "view, browse and play" function that allows you to perform what-if analysis. This feature is great, especially for power users who want to try to second guess the job of relocating device drivers and TSRs Optimize has done.

Me too

In addition to some of the unique memory management problems posed by certain hardware, as software becomes increasingly complex with huge programs supported now by extended memory, there is an accelerating trend toward specialized, often proprietary, memory management as well. This trend is nothing new. Quarterdeck's QEMM and DESQview were a team before Quarterdeck renamed the package DESQview 386. In writing complex programs (DESQview certainly has to fall into that category), it can be a great help if those writers can say, "Gee, if we could do this with memory . . ." and then do it: a tweak here, a diddle there. If you want to squeeze the absolute maximum performance out of DESQview on an 80386, QEMM is the way to go, or QRAM on a 286.

Windows 3.0 is another case in point. It needs some rather special memory handling, as you will see in greater detail in the Windows chapter later on. Here, at least two third-party manager developers, Quarterdeck and Qualitas, already are marketing upgraded versions of their well-known 80386 products that include specific Windows 3.0 support.

There is another side to this story, however, because the third-party support in this case is not as third party as might appear at first glance. According to one story making the rounds, Quarterdeck was the first to figure out some way to make some special allowances for Windows 3.0; however, other stories would have Qualitas in a virtual dead heat with its own tweaks and diddles. At which point, Microsoft apparently said, "Whoa, if we can't stop you guys, at least let us write a driver you can all incorporate." The result of that was a shared driver—apparently available to all developers—based on inside information.

There is still another group of software writers who, for one reason or another, make the memory management needed by their programs integral to the programs in such a way as to preclude the use of any other form of memory management. An example of this approach is seen in VM386, a modular multitasking/multiuser package that is featured in a later chapter.

Exceptions to the rule

From the beginning (once we got away from hardwired boards), memory management has been essentially a software function, limited mainly by the constraints of the hardware—restraints of the basic system and of whatever add-in memory boards might be installed. This was fine for 386 owners, but was of no help to 286 owners stuck with the design deficiencies of that chip.

Eventually, some clever designers figured out a way to get around the limitations of the 286—a hardware solution for a hardware problem. While other companies wrestled with the problem, a Canadian group hit the ground running with something called the ChargeCard. This palm-sized card plugs into the socket for the CPU chip—between the socket and the chip. The ChargeCard's circuitry provided raw memory management potential to the 286, without even taking up a precious expansion slots.

The hardware couldn't do it all; you still had to have a software driver. Here, the ChargeCard's creators made an even bolder move. If DOS was limited to 640K by having the video plopped almost in the middle of DOS's megabyte, they would move the video. They would just pick up that 640K barrier, lock, stock, and barrel, and plop it someplace else.

The immediate result was a supercharged 286 system with as much as 960K of DOS-addressable RAM—as much as 50% more user memory. (It sounds easy, but in practice All encountered problems with revectoring the video and at this writing was no longer supporting this feature. A more recent entry to the 386 memory manager market, Memory Commander from V Communications, has incorporated a similar feature, which I will examine in more detail in the next chapter.) This system, in some ways, was more powerful even than the most powerful 386. It could even run virtually any 386-specific software, at a significantly more attractive price than the cost of an accelerator card or system board swap.

Not content with that, while several other companies raced to develop similar devices of their own, the ChargeCard people at All Computer set their sights not only on using some of the same memory management technology to further enhancing the power of the inherently more powerful 386 bit also to develop another special card to make the advantages of memory mapping available to the 8088.

Add-on hardware support even for the 8088 is feasible, although not at a level comparable to what can be added to an 80286. Unlike the strong upgrade market All found for their 286 card, reaction was less enthusiastic for its 8088 card, which has been withdrawn from the market. As of this writing, there was at least one other company that was still marketing a similar add-on upgrade for the 8088 chip, but the cost effectiveness of any such upgrade is questionable at this point. There are now several similar 286 hardware upgrade options available from Sota and other vendors, which are breathing a lot of new life into lots of 286s.

Shots in the dark

More recently (as mentioned in an earlier chapter), a couple of new memory managers for the balky, unadorned 286 have hit the scene, even offering some help to owners of some better 8088 machines, as well. Again, the major players in the game are Quarterdeck with QRAM and Qualitas with MOVE'M.

Although results vary significantly between machines from different makers, neither of these programs are going to set the world on fire—particularly on an 8088. They are interesting and, at least on 286s, they aren't just dead ends in themselves but can be carried over onto ChargeCard-upgraded systems. Quarterdeck in particular claims better performance than with just the software driver All supplies.

You need to understand that with both of these (and presumably any look-alikes that might emerge from other sources), 286s are not all created equal. Some are surely more equal than others, depending particulary on whose supporting chip set shares the board. The best chip sets to date have come from Chips and Technology and from NEAT (New Enhanced AT). These chip sets will not match a 286 that has been hardware-upgraded with a ChargeCard, but you can expect significantly better performance when mapping memory to the 640K to 1024 area than with most other chips.

Getting downright pushy

Periodically, someone comes up with the idea of revectoring, or moving, the video memory out of its comfortable but inconvenient location. Indeed, there are ways it can be done with existing hardware and applications through clever memory management programming. By relocating the video memory area up just under the EMS page frame (or doing away with the page frame right up underneath the system ROM), everything south of the border could be mapped with RAM.

It is not just RAM, but RAM contiguous to 640K, forming an unbroken chain of as much as 960K of linear addresses. Not only does the scheme allow DOS applications to have more directly accessible space for files and data but, working with one contiguous block (even though TSRs and device drivers would load in conventional memory), they also would load more efficiently than if squeezed, often wastefully, into fragmented upper memory blocks.

There's nothing really sacred about A000h and most of the address space up to about BFFFh, so it's not surprising that that possibility keeps coming up. Unfortunately, it's easier said than done. All Computers, Inc., for instance, at one time supported revectoring and advocated its use. The company has since quietly backed away from it, however. Although it does work, All found it does not work with all software.

More recently, a new third-party 386 memory manager featuring video revectoring has been introduced by yet another company, a relative newcomer to the scene. Aside from some other seeming flaws, it does present an opportunity to reexamine the revectoring issue. On balance, it still doesn't seem to fly.

I was indeed able to get as much as 836K of contiguous memory and still have a 64K page frame (at E000h). That amount does not represent a net gain of the nearly 200K that the raw numbers would indicate, but rather a much smaller figure, as reported in the Manifest printout shown here:

```
Memory Area   Size   Description
2F04 - 3019   4.3K   COMMAND
301A - 301E   0.1K   [Available]
301F - 302F   0.3K   COMMAND Environment
3030 - 3037   0.1K   PZP Environment
3038 - 39C8    38K   PZP
39C9 - 39D0   0.1K   MC Environment
39D1 - 3A12     1K   MC
3A13 - D5FF   623K   [Available]
```

Eliminating some 38K of overhead for a screen grabber (PZP) I use to catch screens to use as illustrations as I go, this information would seemingly yield just over 660K of contiguous DOS space to run an application. This example, however, is a bare configuration, without any of my normal device drivers or TSRs loaded (typically about 50K), which would more than offset that difference.

The net gain in the implementation at best is really not that great, as evidenced by this second report:

```
Memory Area   Size   Description
0D01 - 0E16   4.3K   COMMAND
0E17 - 0E1B   0.1K   [Available]
0E1C - 0E2C   0.3K   COMMAND Environment
0E2D - 0E35   0.1K   [Available]
0E36 - 9FFF   583K   [Available]
= = = =Conventional memory ends at 640K= = = =
```

The report shows that about 35K less of the same usable memory is available on the same machine using a different memory manager (QEMM) with no revectoring and with 50K or so of device drivers and TSRs loaded into upper memory. The key would seem to lie in the system overhead imposed by memory manager (roughly 200K, in this case, before COMMAND.COM even starts to load, as opposed to about 50K in the latter case). In any case, there's a lot more than just the top number that you've got to keep an eye on.

I'll make one final point. I fell back on Manifest rather than capturing the memory usage display screens this program gives because my screen grabber wouldn't work with the video relocated. It would not even work with the contiguous memory reduced to 736K and the graphics memory retained except that everything just moved up a notch. Other strictly text programs ran, but don't pack your bags and move in that direction just yet. It might be time to make a move in some direction, however, because the world is changing fast. In the closed corporations of our world, the simple operating system itself is available with some interesting look-alikes to pick from now; it is these corporate chairmen of the board that hold the power for the direction of change.

10
CHAPTER

Two times 5.0: not quite a ten

As most users know, there are two DOSs to pick from these days: the Microsoft variety and rival DR DOS from Digital Research. The question arises every time a new DOS version is released as to whether an upgrade really is needed. The question now has yet another factor to consider: whose DOS do you choose?

As the DOS wars heat up between these two traditional rivals, the users ultimately will be the real winners. Maybe two times 5.0 did not add up to 10, but this one's not over yet. Regardless of where this finally ends up, users have two powerful contenders even now, both more powerful than anything seen before 5.0.

There are many similarities between the two DOSs—so many that you could pretty much install DR DOS 5.0 on the machines used by your colleagues without their even realizing you had switched their operating system. Until you get into some of the more advanced features (options and switches that did not exist before 5.0), the commands and syntax are essentially identical. They are remarkably identical (even odd-ball things like NSLFUNC are there in both of them).

Because they are so similar in many ways, it would not be fair to limit my discussion here to strictly MS-DOS, especially because, at this point, both offer memory management features formerly available only from such third-party EMS specialists as Quarterdeck and Qualitas. Also, for the first time there is, I feel, a truly credible alternative to MS-DOS—and don't think Microsoft hasn't felt the heat.

DR DOS 5.0 was released a year or more before the new MS-DOS 5.0 and not long before Microsoft, to their credit, began one of the most extensive beta test programs ever launched to assure that the 4.0/4.01 fiasco was not repeated.

During that beta phase, several features introduced by Digital appeared—and sometimes disappeared again—in succeeding MS-DOS beta copies.

As MS-DOS 5.0 finally emerged, it had features that weren't shared by DR DOS 5.0 (although another release of DR DOS, which already has been announced, ups the ante). The reverse also is true. Microsoft focused more on the needs of less experienced users and the Windows market, while Digital's priorities seem aimed more at the power user market with advanced memory management features MS-DOS 5.0 just never quite caught up with, even given an extra year MS-DOS had in which to get its act together in the war of the 5s.

I do not say that to be critical of MS-DOS 5.0 by any means. It is a very good release, coming like a breath of fresh air after 4.*x*. There clearly are some philosophical differences behind the two DOS 5s. These are things that need to be explored at this point, especially as Microsoft puts increasing emphasis on Windows and users are forced to make decisions.

Left foot, right foot

Marching to the beat of different drummers, it is not surprising that the two DOSs start off on quite different footings as far as memory management for 386 and *i*486 machines is concerned. While both provide XMS support for extended memory on 286 machines, only MS-DOS provides a true XMS driver for 386 and higher machines, providing EMS emulation via a secondary driver (EMM386.EXE). DR DOS, on the other hand, takes the more traditional direct LIM 4.0 EMS approach with its EMM386.SYS driver that, unless you tell it otherwise, turns all spare memory into EMS memory.

Although there are certain advantages that can be argued for either way of going about it, to most users any difference there might be is not likely to be noticed. Because Windows 3.0 chose to go the DOS-extender route, that simply put a little higher priority on extended memory at Microsoft. Otherwise, for the average user, it's kind of like which shoe you put on first when you get dressed: a matter of habit rather than reason. However, there are some notable exceptions as you will see a little later in this chapter.

The Microsoft connection

MS-DOS 4.0 offered token support for extended and expanded memory but of too limited a nature to be taken seriously. This time, Microsoft is serious. The future as Microsoft perceives it, surely motivated in no small part by Windows 3.0's insatiable needs, is in extended memory: linear memory continuing from the point where MS-DOS must stop and simply continuing as an unbroken string of addresses.

MS-DOS's XMS driver, HIMEM.SYS, is the key to the entire MS-DOS 5.0 memory management scheme. While it paves the way to extended memory on

80286 and higher machines, it also provides access to the 64K HMA as well, making it available either for its own use or to be used by other software written specifically to use the HMA if it is available.

Not only does this address the needs of Windows 3.0, but extended memory is a whole lot easier to manage than expanded memory, as evidenced by the relative sizes of typical XMS drivers as compared to EMS drivers, the latter typically on the order of 10 times the size of a typical XMS driver. Given that, MS-DOS 5.0 was written to allow the bulk of the kernel, some 47K, to be loaded to the HMA.

To do this on an 80286 or higher requires only the following two lines in the CONFIG.SYS then:

```
DEVICE=C:\DOS\HIMEM.SYS
DOS=HIGH
```

The first line simply loads the XMS driver to provide extended memory and HMA support (assuming you have sufficient RAM installed, although any machine that came with 1 Mb installed has enough for this at least). That line should be at or near the top of the CONFIG.SYS. The next line is simply a loader that tells DOS where to put the kernel (the default is LOW). That line can be almost anywhere in the CONFIG.SYS.

Although the idea of moving the kernel out of conventional memory to make room for bigger applications is hardly new (both PC-MOS and DR DOS did it well ahead of Microsoft), this, from a memory management point of view, is probably the most important new feature in this release. I say that because it is the only memory management issue that cannot and could not be accomplished solely by third-party memory management software.

The MS-DOS 5.0 kernel can be relocated into HMA memory provided by any third-party XMS driver as easily as if HIMEM.SYS was used. However, the kernel cannot move itself but rather requires the presence of a memory manager to provide a place for it. The capability of being moved at all is something that must be specifically written into the kernel—that and how and where it can be moved to. It is critical that you understand this relationship in order to understand important differences between the two DOS 5s.

With an XMS driver capable of providing not only extended memory but a High Memory Area on 80286 and higher systems, Microsoft elected to relocate the MS-DOS kernel to the High Memory Area and only to the High Memory Area, no matter how much spare memory you might have mapped to high DOS address space—space going to waste perhaps—on 386 or higher systems.

Some 47K of the kernel can be relocated to the HMA, provided that no one else is using the HMA. It is winner takes all in that block, unless you have another program that can use more than 73 percent of the available 64K (DESQview can use 63K or 98 percent). You are forced to make a choice, because MS-DOS can relocate the kernel to the HMA only. Either you put the kernel in the HMA or the whole kernel is going to sit down in conventional memory the way it always has.

A cut above

In addition to offering an XMS Extended memory manager in this release, MS-DOS 5.0 for the first time offers expanded memory emulation as well—full LIM 4.0 EMS expanded memory emulation, but only for use on 80386 and higher machines despite the fact that even 8088 PCs and clones can use expanded memory.

Most of those 8088 and even many 80286 machines, however, can only have memory available for nay usage beyond 640K by adding expansion cards, such as the AST RAMpage. These generally provide their own proprietary memory management software. Viewed in this context then, the limiting of expanded memory support to the upper level of machines and catering specifically to features found only on those machines makes sense.

EMS memory is the memory you need for any applications that require expanded memory. It also is the only kind of memory that can be mapped to unused address space above 640K on 386 and *i*486s. To have expanded memory using MS-DOS 5.0s new scheme, however, you first must have extended memory. This is not an option. The new expanded memory emulator, EMM386.EXE, has to find a chunk of extended memory that it can manage in a way that makes you, your computer, and your software think you have expanded memory.

Because EMM386.EXE can work only when it has extended memory to work with, it is critical that HIMEM.SYS be installed ahead of it in this manner:

```
DEVICE=HIMEM.SYS
DEVICE=EMM386.SYS
```

The EMS emulator does not have to be the next item in the CONFIG.SYS. You can slip drivers or CONFIG.SYS-level commands in between, but only if they do not require the use of EMS memory. There is a definite pecking order that must be observed here. You cannot access extended memory until you have installed the XMS driver. The same holds true for any items that will use EMS memory: the driver for EMS memory can function only when it has extended memory already in place to work with. The easiest thing then is simply to put any memory management drivers right at the top of the CONFIG.SYS in whatever sequence they must follow and be done with it.

Just having EMS is not enough

While many programs use expanded memory in one way or another, once you have provided for the memory they need, the rest is up to them. This is not the case when it comes to EMS memory that you want to map to high DOS address space. While EMM386.EXE is capable of finding some unused address space and mapping to it, it can hardly be called aggressive in its efforts.

EMM386.EXE is blind to any empty address space that might exist below C000h or any that might be mappable above F000h. Operating within those constraints on my own machine it can find only 75K in the DOS area that's mappable. However, there are at least two other usable blocks on that machine, which means that if I need more than just that one 75K block—which I always do—I have to force the issue.

Fortunately, like most EMS drivers, EMM386.SYS can be forced. It can be forced to include other blocks that might be usable. It can be forced to ignore blocks it might think are open when you know you have a piece of hardware that has not been recognized that early in the boot process. To do that, you must tell EMM386.EXE what specifically to do as part of its command line in the CONFIG.SYS. For example:

```
DEVICE=C:\DOS\EMM386.EXE 256 RAM I=B000-B7FF i=F800-FDFF
   X=C800-CBFF FRAME=E000
DOS=UMB
```

Here, in addition to the 75K found by default, I've used I= statements to include another 32K starting at B000h, plus 24K of ROM I know I can map over on that machine starting at F800h. A data compression board was sitting at C800h, so the address space it uses was excluded with an X= statement. For some reason, EMM386.SYS wanted to put the EMS page frame at D000h, breaking even the 75K it found on its own into two smaller blocks so the FRAME= statement forces it to use E000h as the starting address instead, keeping the 75K block intact. The DOS=UMB (a separate command on a separate line) must be included to tell DOS what to do with all that space—in this case, to map EMS memory to the available Upper Memory Blocks. This is what you can do with EMM386.EXE in a typical situation, as demonstrated in Fig. 10-1.

```
EMM386 successfully installed

  Available expanded memory . . . . . . . .  512 KB

  LIM/EMS version . . . . . . . . . . . . .  4.0
  Total expanded memory pages . . . . . . .   28
  Available expanded memory pages . . . . .    4
  Total handles . . . . . . . . . . . . . .   64
  Active handles  . . . . . . . . . . . . .    1
  Page frame segment  . . . . . . . . . . . E000 H

  Total upper memory available  . . . . . .  131 KB
  Largest Upper Memory Block available  . .   75 KB
  Upper memory starting address . . . . . . B000 H

EMM386 Active
```

10-1 If you can read really fast, EMM386. EXE quickly scrolls a message similar to this one as it loads. This is only of general interest, however, and of little benefit when you are trying to utilize upper memory.

You've really got to read fast (or freeze the scrolling screen with Ctrl—NumLock) to read that as the system is booting. That opening screen really isn't the information you need anyway. You're better off using the MEM command with the /c (classify) switch.

I loaded some drivers up there so you have something more to look at. Figure 10-2 shows what MEM displays in a typical situation with the kernel loaded to the HMA.

```
C>MEM /c

Conventional Memory :

  Name              Size in Decimal       Size in Hex
-------------     ---------------------   -------------
  MSDOS               15536       ( 15.2K)      3CB0
  HIMEM                1184       (  1.2K)       4A0
  EMM386               8400       (  8.2K)      20D0
  COMMAND              2624       (  2.6K)       A40
  FREE                   64       (  0.1K)        40
  FREE               627344       (612.6K)     99290

Total  FREE :        627408       (612.7K)

Upper Memory :

  Name              Size in Decimal       Size in Hex
-------------     ---------------------   -------------
  SYSTEM             184320       (180.0K)     2D000
  RAMDRIVE             1184       (  1.2K)       4A0
  SMARTDRV            17904       ( 17.5K)      45F0
  EGA                  3280       (  3.2K)       CD0
  STACKER             22336       ( 21.8K)      5740
  MODE                  464       (  0.5K)       1D0
  XYKBD                1552       (  1.5K)       610
  FREE                   48       (  0.0K)        30
  FREE                30880       ( 30.2K)      78A0

Total  FREE :         30928       ( 30.2K)

Total bytes available to programs (Conventional+Upper) :     658336
(642.9K)
Largest executable program size :                            627216
(612.5K)
Largest available upper memory block :                        30880
( 30.2K)

    458752 bytes total EMS memory
     65536 bytes free EMS memory

   3407872 bytes total contiguous extended memory
         0 bytes available contiguous extended memory
   1986560 bytes available XMS memory
           MS-DOS resident in High Memory Area
```

10-2 A MEM report with the /c (classify) switch showing over 612K of conventional memory with the kernel loaded in HMA, half a dozen device drivers loading into upper memory, and space still remaining for more. However, this does not show the addresses where specific drivers are loaded. That information requires running MEM again with the /d switch.

What this figure shows is that I've loaded all four of those device drivers plus two small TSRs (from the AUTOEXEC.BAT) into upper memory blocks. There still is over 30K that can be used up there. That's not bad for an afternoon's work, and even better considering it only took minutes to do—just long enough to add the EMM386.EXE driver and the UMB, to change all of the DEVICE= statements to DEVICEHIGH= statement and to reboot.

This was a fairly easy case. By the same token it was fairly typical and clearly well within the capabilities of anyone who can write and/or edit a CONFIG.SYS. Even without getting fancy, there should be enough power here for many users—probably for some time to come.

Still, they all won't go this easily. There are some nasty programs out there that require a lot more memory during loading than they need to run. To make things fit, you sometimes have to juggle things around. Sometimes you have to change the loading order by changing the order things are called for in the CONFIG.SYS (or AUTOEXEC.BAT for TSRs). Sometimes when all else fails, you simply have to let the overflow load low, which then comes down to trying to squeeze the most bytes into upper memory, so you have the fewest spilling over if you want to do it right.

When you get into having multiple blocks of mapped memory to work with, working out a loading sequence that packs the most in becomes much more of a problem. MEM used with the /d (or /debug) switch can help by providing specific address information as shown below (left hand column):

```
01E4D0   MSDOS        081B10   -- Free --
09FFF0   SYSTEM       02D010   System Program
0CD010   IO           00AEE0   System Data
           RAMDRIVE   0004A0    DEVICE=
             E:                   Installed Device Driver
           SMARTDRV   0045F0    DEVICE=
             SMARTAAR             Installed Device Driver
           EGA        000CD0    DEVICE=
             EGA$                 Installed Device Driver
           STACKER    005740    DEVICE=
             F:                   Installed Device Driver
0D7F00   MSDOS        000030   -- Free --
0D7F40   MODE         0001D0   Program
0D8120   XYKBD        000040   Environment
0D8170   XYKBD        0005D0   Program
0D8750   MSDOS        0078A0   -- Free --
```

This however gives no indication of the space required to initialize (to load) any of these drivers, which is often significantly larger than the space to run them once they settle in. Just because the raw numbers seem to indicate that you've got the space to load another one doesn't mean that you can make it happen, even if you try a lot of different loading sequence combinations. This problem was discussed in an earlier chapter; however, because of its importance, I will discuss it again later in this chapter.

Despite the fact that MS-DOS 5.0 lacks amenities provided by some of the better third-party memory managers—amenities that make life easier above 640K—the raw power is there. I was pleasantly surprised how well the HIMEM .SYS/EMM386.SYS duo works with only just a little help.

From digital research: DR DOS 5.0

Outwardly DR DOS 5.0 is so close to being a perfect clone for MS-DOS that you could probably slip it over on most of your friends or colleagues without their even knowing. DR DOS 5.0, however, is unique in many ways. Derived from a different ancestry, it is philosophically quite different. Nowhere is this more apparent than in the area of memory management.

Much of what was said about MS-DOS 5.0 applies equally—or nearly so—to DR DOS 5.0. I'm going to cut right to the chase and focus on the primary areas in which they differ in the way they manage memory, pointing to specific differences when and where they're really relevant.

First off, I need to define the fundamental underlying difference between the way these two DOS 5.0s manage money. Microsoft, as pointed out above, has put its emphasis on extended memory, using an XMS driver as the cornerstone of its entire memory management strategy. That XMS driver, in addition to providing extended memory, supports a 64K High Memory Area (that extra 64K DOS can access in real mode although it lies beyond 1 Mb, making that HMA available as an alternative to loading the bulk of the DOS kernel in conventional memory).

Digital, as stated earlier, has gone a different route, making EMS memory its primary concern for 386 and higher systems. Digital provides an XMS-compliant driver only for the 80286 user base to whom EMS memory mapping to address space above 640K is an option except in special situations that their HIDOS.SYS driver can accommodate.

Digital also has written the kernel in a way that allows a good part of its bulk to be relocated above 640K. On an 80386, it can be given a starting address right at the top of DOSs 1 Mb of address space and can be loaded into the address space that, with an XMS driver, is the High Memory Area. DR DOS's 386 memory manager, EMM386.SYS, is not an XMS driver, so the kernel is loaded into that never-never land just above 1 Mb. However, it isn't in the High Memory Area because EMM386.SYS is not an XMS-compliant driver and, therefore, does not support the HMA.

More than just a matter of semantics

The fact that DR DOS's EMM386.SYS is not a true XMS driver despite the fact that it can relocate the bulk of the Digital kernel just above 1024K in an area which is the High Memory Area as defined by the XMS specification is more than simply a matter of semantics. Under the industry-standard XMS protocols, the area between 1024K and 1088K can be accessed by any software adhering to those protocols (it's called HMA-aware). As discussed earlier, it is up to the user to decide which, if any, of his or her programs get to have the HMA. However, Digital's 386 driver does not abide by the XMS protocols for access to this area; therefore, despite the fact that DR DOS can use that address range for its own purposes, it cannot make that area available to other programs that are HMA-aware.

This phenomenon is not unique to DR DOS, but is common to several other packages including PC-MOS, one of the multiuser multitaskers I'll discuss in the next chapter. That address space is not wasted if the kernel is not relocated above 1024K, but is normally available to other software only as extended memory.

On the other hand, by going the route Digital has chosen, it has not limited itself to only the space above 1024K to relocate its kernel as Microsoft has done. With DR DOS 5.0, you can, at your discretion, relocate the kernel to any contiguous address block above 640K that has a minimum of something like 38K or EMS memory mapped to it.

In its next release, Digital has taken this still farther, breaking the DOS kernel into several smaller modules that can be loaded all together, if there is a block sufficiently large, or separately in smaller blocks. It just doesn't get much better with a real mode operating system.

This opens up a number of possibilities, not the least of which is using one of Digital's memory management utilities (HIDOS.SYS) in combination with a third-party driver like Quarterdeck's QEMM, which does provide XMS support for a High Memory Area. Although this does not seem to be documented in the user's guide, it is detailed in supplemental release notes and is not only a viable option but a highly desirable one for DESQview and many users with specific memory management problems that involve a combination of HMA and upper memory usage.

Because Digital's EMM386.SYS memory management scheme for 80386 and higher systems revolves around the use of EMS memory, Digital provides, out of a necessity, a separate drive, HIDOS.SYS, for 80286s. This one, unlike Digital's 386 driver, is an XMS driver that supports the use of the High Memory Area as a distinct entity.

HIDOS.SYS not only provides extended memory and HMA support but, when used with supporting hardware, can make upper memory available on 286s. Here, however, supporting hardware is the key and, as of this writing, only computers using Chips and Technologies LeAPSet, LeAPSetsx, NEAT, and NEATsx chip sets provide the level of support required. In the next DR DOS release to follow 5.0, this has been extended to support still other chip sets. This is something MS-DOS does not address at all at this point, nor is it likely to in the foreseeable future, barring a total overhaul of HIMEM.SYS and its total strategy.

Interestingly, HIDOS.SYS is also the management utility that can be teamed with third-party memory managers like QEMM or 386MX, making it possible to put 38K of the DR DOS kernel pretty much where you want it above 640K. Here, it does not perform a memory management role per se but rather, under the auspices of a third-party memory manager, serves only as a loader, managing the relocation of the DR DOS kernel into memory provided by the other driver.

To put this in perspective, using this technique and combining Quarterdeck's QEMM.SYS with HIDOS.SYS, DR DOS gave me nearly 600K of contiguous conventional memory to run applications—598K to be exact. I still had another 113K mapped to upper memory blocks, plus the HMA to work with, which is more than enough to satisfy the needs of DESQview plus another driver or two. Because I rarely run in graphics mode, I could have grabbed video graphics block between A000h and AFFFh to add still another 64K to the pot if I had needed it.

Some people might argue that introducing a third-party memory manager is not fair when comparing the relative merits of DR DOS 5.0 and MS-DOS 5.0. Regardless of what it took to do it with DR DOS, this is performance I have been unable to match during months of working with MS-DOS 5.0. Fair or not, I rest my case.

Additionally, DR DOS provides a special EMS driver, EMMXMA.SYS, for use with memory cards compatible with IBM XMA memory expansion cards. This driver maps memory to a 64K window within the range from C000h to DFFFh and can be assigned only a fixed portion of the available memory resources to use as LIM 4.0 EMS memory or, if no limit is specified, can take whatever it can get.

To load TSRs, drivers, and the DR DOS kernel above 640K, Digital supports a command set, which, with the exception of HINSTALL, is similar to those provided by Microsoft for MS-DOS. All of them default to loading in conventional memory when insufficient upper memory is available. They include:

HIDOS=ON/OFF	Used in CONFIG.SYS to load the kernel high or to force it low.
HIDEVICE=	Used instead of DEVICE= in CONFIG.SYS to load the drivers to upper memory
HINSTALL=	Similar to the INSTALL command in both DOSs, except that it loads supported TSRs to upper memory from CONFIG.SYS
HILOAD	Used from the command line of a batch file to load TSRs to upper memory

Although the names might differ slightly from their MS-DOS counterparts, they function similarly, except for HINSTALL.

HINSTALL is the exception, differing from INSTALL only in that, when run from the CONFIG.SYS rather than from the AUTOEXEC.BAT or command line, it loads TSR's above 640K rather than in conventional memory. INSTALL, in my opinion, is generally counterproductive in either DOS, with the emphasis these days on loading things above 640K whenever possible. DR DOS's HINSTALL which does not have a MS-DOS counterpart, is very interesting, doubly so when combined with the multiple CONFIG.SYS configurations options supported by DR DOS.

Configuring your system on the fly

One of the neatest options DR DOS adds to the equation has nothing to do with memory management per se; it just makes life a whole lot easier as far as optimizing your configuration is concerned. Digital has endowed the CONFIG.SYS with batch file-like branching capabilities, even adding an extra feature that will pause and prompt for input as it runs. This allows the option to choose between several different configurations and, within each configuration, to choose which device drivers you load for that particular session.

Microsoft really missed the boat when it failed to pick up on this one, because different configurations can even include installing different memory managers. You can use 386MAX with its instancing feature and whatever else you might want specifically for a Windows session, use QEMM for DESQview, use still another configuration to use DR DOS's own memory management tools, or use a bare configuration for a VM386 session. Because it is so like batch file programming, anyone who can write a batch file can use it. A typical DR DOS CONFIG.SYS might look something like the model shown in Fig. 10-3.

In addition to the flexibility this affords once you have your system fully configured, it is a great development tool when fine-tuning your system. You can leave your existing configuration intact but still experiment as much as you like without ever losing the ability to return to your old, proven configuration simply by rebooting.

Choices, choices

So there are not one, but two DOS 5.0s, coming from the two arch rivals that first squared off against each other for the early PC market. In their primary function as disk operating systems, both look pretty good. I would be hard put to make an irrevocable commitment to either over the other. In terms of actual memory management beyond 640K (which is what this book is all about), however, after working extensively with both, I have to give a definite edge to Digital for offering more options, particularly when it comes to relocating a big chunk of the kernel.

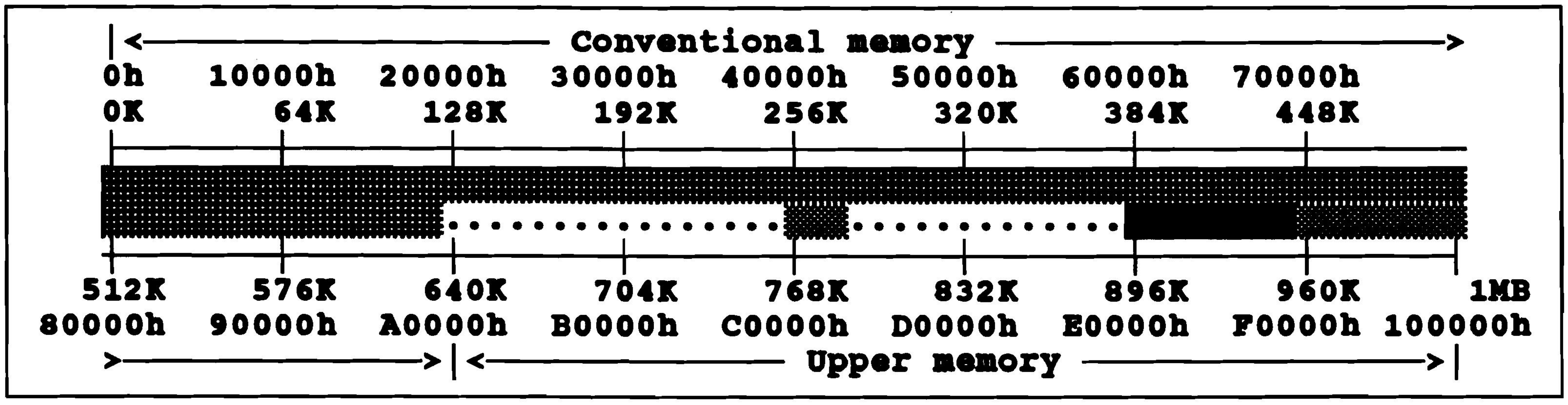
```
?"Do you want to run standard configuration?" GOTO STANDARD
?"Do you want to run DESQview?" GOTO DESQview
?"Do you want to run a WINDOWS session?" GOTO WIN
:STANDARD
?device=a:\drdos\emm386.sys /BDOS=FFFF /F=E000 /AUTOSCAN=A000-FFFF
  /KB=1024 /USE=F800-FDFF /EXCLUDE=C800-CCFF
GOTO COMMON
:QEMM
DEVICE=C:\DOS\QEMM\QEMM386.SYS RAM ROM EXCLUDE=C800-CCFF AU DMA=32
DEVICE=A:\DOS\HIDOS.SYS /BDOS=AUTO
HIDEVICE=device
HINSTALL=TSR_program
GOTO COMMON
:WIN
•
•

:COMMON
?HIDEVICE=....
```

10-3 DR DOS supports a prompting feature and batch file-like branching that allows users to select from customized configurations when booting. The question mark, which causes DR DOS to pause and prompt for input, can preface branching commands or individual loading options. Note how optional configurations can include both device drivers and TSRs not common to any other configuration.

Also, DR DOS's HIDOS.SYS XMS driver provides special support for computers using Chips and Technologies LeAPSet, LeAPSetsx, NEAT, and NEATsx chip sets— something not provided by MS-DOS's flagship XMS driver. Additionally, I think DR DOS's MEM, with its graphic presentation (Fig. 10-4) and greater number of display options, runs rings around MS-DOS's MEM.

Otherwise, I have pretty much found that, if you can do it on a 386 with MS-DOS 5.0's HIMEM.SYS plus EMM386.EXE, you can do it with DR DOS 5.0's


|<— Conventional memory —>
0h 10000h 20000h 30000h 40000h 50000h 60000h 70000h
0K 64K 128K 192K 256K 320K 384K 448K
512K 576K 640K 704K 768K 832K 896K 960K 1MB
80000h 90000h A0000h B0000h C0000h D0000h E0000h F0000h 100000h
>—>|<— Upper memory —>|

```
Key:  ▓=RAM  ▒=ROM  ⁞=Shadow ROM  █=EMS

      655,360 bytes, ( 640K), conventional memory
      587,488 bytes, ( 573K), largest available block

    3,407,872 bytes, (3328K), extended memory
    3,407,872 bytes, (3328K), extended memory used
            0 bytes, (   0K), extended memory available
```

10-4 A portion of a DR DOS MEM display. Six switches (plus a help switch) allow data to be displayed in a tabular form that is more concise and generally easier to interpret than MS-DOS MEM data.

EMM386.SYS. Having said that, however, both of them are running just about two years behind the leanest, meanest third-party competition in the memory sweepstakes: most notably Quarterdeck, Qualitas, and more recently All Computer's 286 and 386 memory management offerings.

Singularly lacking in both DOS 5s are tools like QEMM's and QRAM's automatic memory Optimize utility and Manifest or 386MAX's Maximize. These two were joined recently by all Computers, which now provide utilities to automatically optimize memory usage with both its 286 and 386/*i*486 memory management packages. This would be my biggest complaint with both of them.

For the power user experienced in working above 640K, the basic tools that the utilities provide are good—not great, but certainly adequate to meet the needs of most 386, *i*486, or even 286 users. In both cases, however, that falls considerably short of making it happen. Ironically, the better you are when it comes to working above 640K, the more this becomes a problem.

Don't send a boy to do a man's work

This shortcoming comes back to the same problem I addressed in another chapter, which essentially comes down to the simple fact that no 386 memory management scheme is really any better than its ability to help you locate address blocks that can be mapped. This is especially true when mapping over ROM addresses, which is something few users would even attempt without some guidance—few even with guidance for that matter—with an extra 24K or even more located up there that can be used sometimes, it is surely worth the effort.

Ironically, the better you become at sniffing out the last possible blocks of mappable address space, the more impossible it is in many cases to develop a successful loading strategy by trial and error, which is the only method available with either of the 5s. If your needs for upper memory are modest—say only two or three device drivers and maybe a TSR—and, as one of the cases cited above, the driver by default only finds a single contiguous block of mappable address space, the task is generally pretty easy.

Compound the problem by fragmenting upper memory and then adding to that the fact that many drivers and TSRs require a bigger block to load to than they need to run while others might not, you then get into a sequencing situation. The chart in Fig. 10-5 demonstrates the astronomical numbers of possible loading combinations you can run into—more than you could possibly try if you did nothing else for several years, in some instances.

Even with mappable upper memory fragmented into only three noncontiguous blocks—as in several real life examples used earlier in this chapter—if you have a total of six TSRs and/or device drivers that you want to load up there, there are nearly 1300 possible ways to try to make them fit. Even assuming you have enough total memory available, you can be pretty sure only a few combinations—

		NUMBER OF SEPARATE UPPER MEMORY BLOCKS MAPPED			
		3	4	5	6
NUMBER	3	162	384	750	1,296
OF	4	1,944	6,144	15,000	31,104
RESIDENT	5	29,160	122,880	375,000	933,120
PROGRAMS	6	524,880	2,949,120	11,250,000	33,592,320

Qualitas

10-5 As the number of TSRs and drivers loaded in upper memory multiplies and the number of separate blocks increases, the number of possible loading combinations becomes astronomical, making it virtually impossible to maximize the use of upper memory without the kind of special optimizing software furnished only by better third-party memory managers from Quarterdeck, Qualitas, and ALL Computer.

possibly only one or maybe even none—will actually make them all fit in like pieces of a puzzle.

Admittedly, for the power user experienced in working above 640K, the basic tools provided by either of the DOS 5.0s are good—not great, but probably adequate to meet the needs of most 386, *i*486, or even 286 users. However, in both cases, that falls considerably short of making it happen. This is where you could really use some help—a lot of help perhaps, especially if you're not used to dealing with this kind of problem. This kind of help is something neither DOS 5.0 provides.

Pick either DOS as a basic operating platform that best suits you. You will surely learn from the experience. However, with either of these DOS 5.0s, if you really want to score a 10 in managing and using whatever memory resources you have—or expect to have—beyond 640K, you'd better look beyond just what either of these DOSs have to offer when it comes to really putting memory above 640K to work.

11

CHAPTER

Entry level answers

Many of the ways of unlocking the power beyond 640K are only remedies for users with sophisticated hardware—something better than an 8088 at least—and increasingly suited for nothing short of 386s it seems. Yet there are many ways that even 8088s can perform in ways that far exceed the wildest dreams of the creators of the original PC and do so on a daily basis.

Remember, it was the 8088 that first made expanded memory a fact of life. (Conceptually expanded memory dates back to main frame technology and was used by Apple prior to 8088 PC, but this was the point at which it entered on the PC scene.) Because of the tremendous installed base of 8088 machines—and the continuing market for such machines—there are many options open at this level. In this chapter, I will explore some of the ways of breaking the 640K barrier with even the most modest of machines. However, these are not strictly entry level answers—anymore than DESQview or Windows are exclusive to the high-end market. So these are, to varying degrees, all valid in conjunction even with the hottest and most powerful machines available today; however, all are suited to the 8088 environment.

Some of these involve the use of add-on/add-in hardware: memory expansion boards, external printer buffers and spoolers, modems, FAX boards, and the like. Others, like task switching in its various forms, are largely software solutions and, as such, are available to almost anyone at any level. So this is where I'll start.

Other than the DOS extender, the ability to perform more than one task concurrently on a computer has been one of the greatest breakthroughs—and one of the greatest boons to productivity—since the introduction of a practical desktop computer itself. However, even what is generally called—and will consider in the context of this book to be—true multitasking, is only an illusion—a conjurer's trick. It should come as no surprise then that there are different levels of creating that illusion.

More room with HeadRoom

Initially introduced as offering a way to load as many TSRs as you wanted without using up a lot of precious memory—on the order of what can be done with Pop Drop—HeadRoom has matured to become a unique tool capable not only of playing "now you see it, now you don't" with TSRs but with applications, too.

This is not a multitasker in the sense that DESQview or Windows multitask (i.e., actually being able to keep multiple programs running concurrently). It is a task switcher that is so powerful that it can actually pop up a big TSR—any TSR you happen to have loaded—in the middle of any application you have loaded, regardless of the total memory the combination of those two programs might require.

Let me say that once again: with HeadRoom you can pop up a big TSR—any TSR you happen to have loaded—in the middle of any application you have loaded, regardless of the total memory the combination of those two programs might require, even if that total exceeds the size of the largest single program you can run in the space left over after loading DOS and HeadRoom (HeadRoom itself requires about 50K). This presumes that each program can run independently in something under 640K. In other words, each program has to be able to load and run on just an ordinary PC under DOS.

It's all an illusion. Even though it might appear on the screen as if HeadRoom has popped your favorite whooping TSR up in the middle of your most whopping program, the program code—enough at least to make room for the TSR code—has been swapped out and stays swapped out until you're ready to exit from your TSR, so HeadRoom can swap your application code and/or data back into conventional memory again. Actually, HeadRoom doesn't have to be fussy even about which blocks of foreground code or data it swaps out to make room for a TSR you've called for, because the application is suspended while the TSR is popped up anyway.

At no time do you actually have more real memory than you started with. Sometimes you would swear you do, however, particularly if you've got a fair amount of expanded or extended memory available.

By default, HeadRoom looks first for expanded memory to work with. Failing that, the next best bet is extended memory. If all else fails, HeadRoom has the capability to swap things to and from a disk—but only as a last resort. This is in marked contrast to MS-DOS's new task swapper (part of the MS-DOS 5.0 DOSSHELL), which can swap only to disk no matter what else is available.

One of the nicest features for anyone who doesn't have enough real memory to go around is that HeadRoom allows you to specify the preferred swap medium: extended or expanded memory or to a disk. This allows you to better manage what resources you have, saving precious expanded or extended memory for only those applications and TSRs you use most where disk swapping would cause the greatest inconvenience.

One of the significant features about HeadRoom is that it can swap out programs—even some of the particularly older TSRs—that refuse to run in upper memory on 386s. It gets them out of the way—out of conventional memory—just as effectively as if they normally would, but it never actually runs them anyplace except in conventional memory where everybody's happy. You really can have your cake and eat it, too. I'm not advocating the use of HeadRoom as the ideal 386 solution; however, it is interesting.

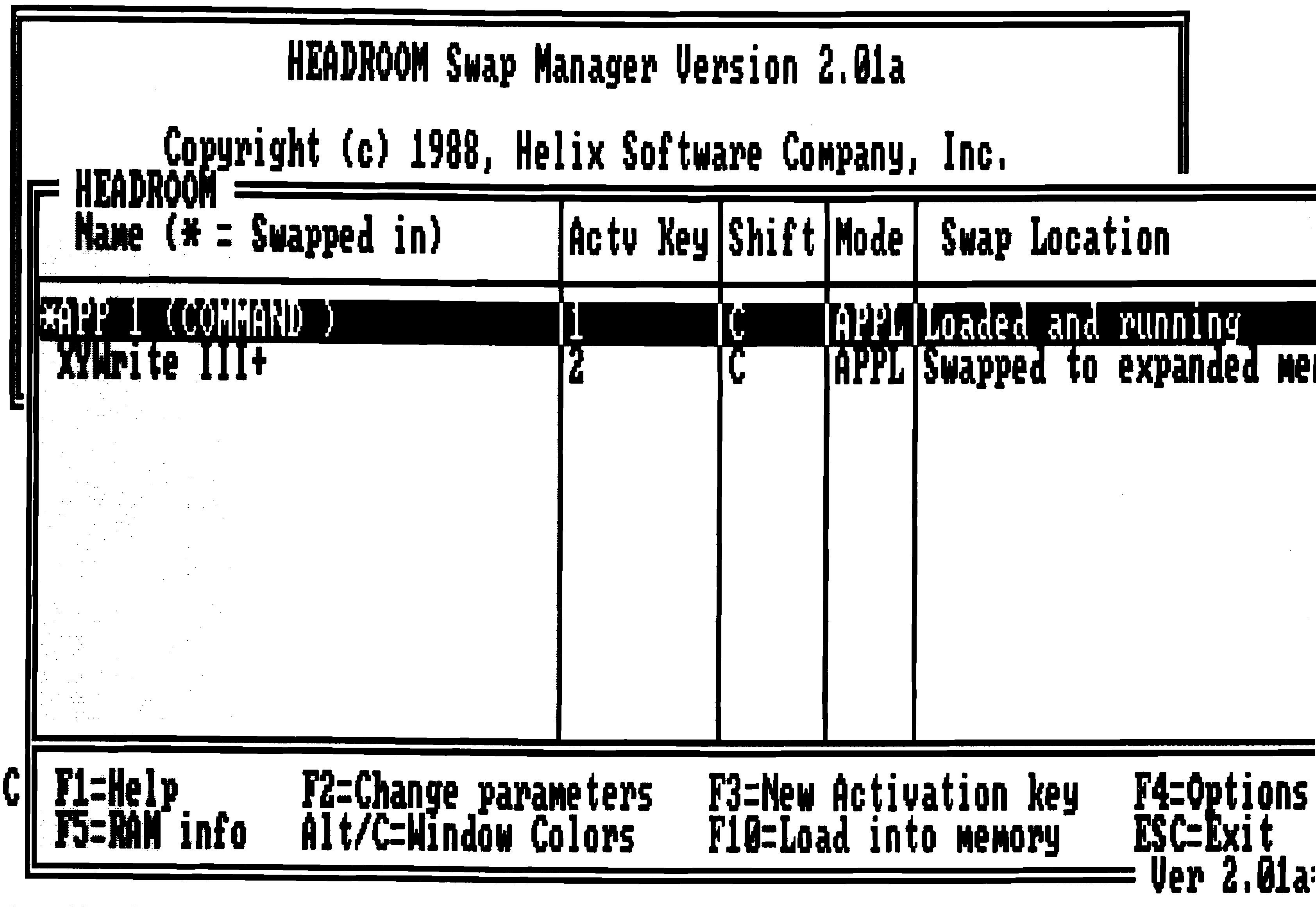
Another interesting feature is that HeadRoom allows programs TSRs of under 64K in size to be loaded directly into the EMS page frame address area. This, it is claimed, speeds access to any such TSR by as much as 30 times, presumably based on disk swap times. This also presumes that the page frame is not needed to access expanded memory being used by other TSRs or applications—in which case, anything loaded into the frame must be swapped out of the way. However, certainly the more direct the access the more immediate the response. To load specifically to this area, HeadRoom provides a special loader called XRUN.

HeadRoom also supports the loading of device drivers into high DOS memory on any machine that supports memory mapping to addresses above 640K; however, this feature is not available on 8088 machines. To accomplish this, two proprietary device drivers are supplied: CDEVSWAP.SYS for character device drivers (printers, display, keyboards and pointing devices, etc.) and BDEVSWAP .SYS for block device drivers (typically nonstandard storage devices). You have to exercise a little judgement here, however, because the device driver swappers are larger than many device drivers (VDISK.SYS or ANSI.SYS for example) to a point that really makes it impractical. Still, it can be done, which makes you wonder why this kind of thing hasn't been exploited by other software developers.

This one is a little trickier to set up than Carousel, for example. The documentation leaves a lot of unanswered questions that could baffle an entry-level or even a reasonably experienced user. As shown in Fig. 11-1, it does have a fairly extensive menu system and on-screen help available, but not enough to meet the need. However, Helix prides itself on having a fine and readily available technical support staff (accessible toll free), but I have some reservations about the ability of some users to know what questions to ask.

In its use of expanded memory, HeadRoom fully supports the LIM 4.0 EMS specification—something Software Carousel does not do at this time.

On the debit side, HeadRoom does not allow you to adjust the amount of memory that each application receives so that you can create smaller windows for



```
HEADROOM Swap Manager Version 2.01a

Copyright (c) 1988, Helix Software Company, Inc.
HEADROOM
Name (* = Swapped in)   | Actv Key | Shift | Mode | Swap Location
*APP 1 (COMMAND )       | 1        | C     | APPL | Loaded and running
XYWrite III+            | 2        | C     | APPL | Swapped to expanded me

F1=Help      F2=Change parameters   F3=New Activation key   F4=Options
F5=RAM info  Alt/C=Window Colors    F10=Load into memory    ESC=Exit
                                                            Ver 2.01a
```

11-1 HeadRoom no longer simply manages TSRs, but now swaps applications in and out of expanded, extended virtual memory as well. However, this is task switching rather than true multitasking.

smaller applications to make what memory you have go further—something Carousel has always done. However, as mentioned earlier, it does allow you the option of always swapping certain applications (and/or TSRs) to disk to save what RAM you have for your most important tasks. Also, at this writing, HeadRoom cannot be loaded above 640K on machines that support memory mapping.

There are some specific warnings attached to using HeadRoom:

- Any disk caching programs must be installed ahead of HeadRoom. This also is true of Carousel, DESQview, Windows, or even individual applications.
- Helix specifically recommends against the use of Microsoft's HIMEM.SYS memory manager.
- HeadRoom does not support Windows/386 or Windows 3.0, except in real mode and on 286s.
- HeadRoom does not support DESQview 386; however other versions of DESQview are supported on 8088 and 80286 machines. HeadRoom also will run nicely under QEMM on 386 machines.

Overall, HeadRoom has to be one of the hottest entry-level options going at this point and close enough to multitasking that it just might serve your needs for quite some time to come.

Context switching: Carousel's revolving door approach

Software Carousel from SoftLogic Solutions was one of the first task switchers to attract a following and to carve out a legitimate niche in the marketplace. It was with Carousel that I first enjoyed the benefits of task switching on an old PC. Today's Carousel is a far cry from the Carousel of yore, however.

More conservative in its approach than HeadRoom, through the years, this venerable product has quietly matured in subsequent releases to a point that, beginning with release 5.0, Carousel has even supported two displays displaying separate applications simultaneously. For instance, you can have your spreadsheet data in front of you on one screen for reference while writing a report using a separate monitor for your word processor. In a more typical configuration, you can have up to about a dozen applications loaded simultaneously, each with its own files open, switching back and forth between them with a simple keystroke combination of your choice.

Now, vying for the 386 market, the Carousel program itself—some 60K—now can be loaded above 640K, which, when run under MS-DOS 5.0 with the DOS kernel relocated to the HMA, allows for windows of up to about 619K—about the biggest you can ever hope to run under DOS. A number of other new features have been added to the current 5.*x* release including mouse support and a more sophisticated printer buffer.

Improved support has been added for FAX or modem communications. Now, programs—typically communications packages—can be preset to load and run automatically at preset times, in addition to a number of other clock/timer features. Carousel also has added a slick new cut-and-paste facility with formatting capabilities that facilitate the transfer of data between different types of applications (groups of spreadsheet cells into a word processing document, for example). For programmers, there is a new API that allows specific Carousel interfaces to be written in either C or Assembler. It comes with a library of ready-to-use C routines.

As the name Carousel suggests, the trick involves exchanging the whole contents of entire blocks of EMS memory—up to a maximum of about 544K each, depending on system overhead. The swap involves not only whatever applications package and any TSRs that you've loaded (once you were inside of Carousel) into that window but also whatever files are opened with them, plus the contents of video memory, so Carousel can give you back the screen exactly as you left it.

By preference, Carousel uses conventional memory first, then looks to EMS (expanded) memory, if it is available. However, it carries with it the ability to use virtual memory (swap to disk) when all else fails. In that regard, if you've got the chips to support it, Carousel can manage up to 9984K of EMS 3.2 or better RAM. This is one of the most significant areas where both Carousel and HeadRoom run rings around the new MS-DOS 5.0 task swapper, which, as I will discuss a little later in this chapter, can swap only to disk, which Carousel also can do for about another 10 Mb.

In any case, however, Carousel uses resources only up to selected limits set by the user, which can be only a part of those available. However, in its ability to use whatever resources are available, Carousel falls into the category of being a hybrid—as are several other software packages discussed in this book.

It should be noted here that unlike DESQview or Windows, Carousel does not and cannot swap actual blocks of memory in and out and continue running code for background applications. Carousel uses an entirely different mechanism—different even than HeadRoom. It does not swap, not in the same sense that the term applies to 4.0 EMS functionality, where the memory and its contents stay where they are and only the addresses are swapped. Carousel meticulously copies the contents of conventional and video memory, block-by-block into whatever storage space you have assigned it. Then, block by block, it reads back in any other previously loaded application you might call for next (or loads it in if it is not already loaded).

If you're using your hard disk to simulate expanded memory instead of RAM, you can watch the little light blink on and off as the disk swallows block after block of data. Either way, even if you have enough RAM to support all this in physical memory, Carousel still has to go through the copy process. It's much faster with RAM than copy/swapping to disk, but RAM is not instantaneous either, typically taking one to three seconds in a 4.77 MHz PC/clone, depending on the size of the block or window being swapped. Compare this to as much as 45 seconds or so to double-swap a large window to or from a disk on the same machine.

It is slow, especially if you're short of RAM and have to wait while Carousel copies back and forth to or from the disk. However, slow is a relative term. Compared to closing files and quitting one application, then loading another, opening files, and trying to find the place you left off last time, slow really isn't all that bad.

Task swapping under the DOSs

Version 5.0 of MS-DOS introduced a task swapping feature, which, as the term implies, supports swapping tasks in and out of the foreground. Normally, because this now is part of MS-DOS, I would have put this first, before HeadRoom or Software Carousel; however, this one, unfortunately, isn't even in the same league.

Both HeadRoom and Carousel swap tasks to EMS memory as long as there is enough of that to go around, swapping to disk only when there's nothing else. The MS-DOS SHELL task swapper swaps only to disk, which makes some kind of sense when you realize the MS-DOS doesn't even recognize EMS memory, providing EMS emulation on 386 and higher machines. So, unless you have a big RAM disk, this one is slow by nature.

On a genuine old IBM PC with a vintage hard disk, swapping typically takes about 25 seconds. Fortunately, while no speed demon at best, given a faster hard disk on a faster machine, the time required to swap one out and another in is reduced considerably. Put in perspective, these swap times are about on par with what you might expect with a context switching program like Software Carousel or even a multitasker like DESQview, if it is not supported by sufficient real memory to handle everything in RAM.

The MS-DOS swapper also imposes considerable overhead on your hard disk—something that could be a problem for anyone like me who's always squeezed for hard disk space anyway. You might not even be aware of it until the day you suddenly run out of disk space in the middle of a session.

It also lacks many of the other amenities found in Carousel or HeadRoom (the ability to preconfigure it to automatically open a selected group of applications, for example). Being only one of several functions provided by the MS-DOS SHELL, the DOS shell also provides a number of services not found in Carousel, so a lot depends on your priorities. Speaking strictly as a task swapper, however, this one leaves a lot to be desired, other than possibly as a set of training wheels for MS-DOS 5.0 users not yet acquainted with the benefits of task swapping.

Do not despair, however. Not to be outdone as the DOS wars heat up, shortly after the release of MS-DOS 5.0, Digital Research announced yet another new release of its own, which added task swapping to the DR DOS repertoire. Although it is not a full-fledged multitasker as incorporated into DR Multiuser DOS (which began shipping in the spring of 1991), Digital's task swapper is designed to use EMS memory, swapping to disk only as a last resort.

Interestingly, Digital is actually in a better position to do this than Microsoft, because, unlike MS-DOS 5.0, the memory management introduced by Digital in DR DOS 5.0—and now enhanced still further—provides EMS emutation, not only for 386 and higher systems but also for an increasing number of the better 80286s that support memory mapping to varying degrees, depending on which chip they are designed around. In this area, Microsoft, by putting its emphasis so strongly on extended rather than expanded memory would seem to a large extent to have excluded that portion of the market that might benefit most from having a decent task switcher.

You might not find some of the features found in Carousel or HeadRoom in Digitals new swapstakes entry. In any event, however, for anyone who's never used a task switcher, there never has been a better time to get your feet wet with all the choices there are to pick from now. Once you start, you'll probably wonder how you've gotten along this long without it.

Task switching lurking in the background

Another approach—and one that allows at least some of the essential elements of multitasking even at the most elemental PC level—is a class of TSRs that sit quietly unnoticed in the background, but take command when activated by some external signal. In a machine that has no real multitasking capability, they take command at that point, simply putting whatever operation you are running in the foreground on hold temporarily, then returning you to whatever point you left off when they are finished.

Communications—via either FAX board or modem—represents one of the best and most popular examples of this kind of background technology. Depending on how much of the time your background operation takes control of your computer away from you and leaves you twiddling your thumbs, this can be a valuable tool.

One of the nicest things about this approach is that, with the exception of some network-like schemes (DeskLink from Traveling Software cannot be used in conjunction with DESQview as both demand access to the CPU via the same channels), you can move right up into a full multitasking environment later without being obsoleted if and when you decide to upgrade your system. So, I'll look here at just some of the typical applications that fall into this category.

FAX boards

FAX is more than just a method of communication. It has become a way of life and a status symbol especially in the corporate world. So these neat little FAX phones are everywhere these days it seems. There are things an ordinary FAX phone can't do that a computer can, so the marriage of the two technologies was natural. Even if you don't need all of the fancy features a computerized FAX has to offer, you can add a FAX board to your existing computer system at about half the cost of even a low-end dedicated FAX phone—a number that has stayed fairly constant even as the prices for both have plummeted.

Using your computer as a FAX transmitter, you can send files right from disk—no scanner or other reader is required. Incoming traffic can either be saved to disk, allowing either temporary or permanent storage, or be sent directly to your dot matrix or laser printer, giving you a crisp copy that won't fade with age or long term exposure to heat and light.

There are really two distinct classes of FAX boards on the market, generally with equally distinctive prices. At the low end of the scale are basic FAX boards—simply modems geared to the specifics of FAX communications. Boards of this type are sold by various manufacturers, not surprisingly including Hayes, which acquired the JTFAX line from Quadram. Such boards typically have proprietary software that can be loaded in the background as TSRs and, if you have a dedicated line, will answer incoming calls automatically. This freezes whatever application you might be running in the foreground at the time until the transmission is ended and the data is safely saved to disk. At that point, the pop-up pops back out of sight, returning you to the application you were running until the next time the phone rings.

These, however, are adequate for the needs of many users, particularly where little FAX traffic is anticipated or when the bulk of whatever traffic there is can be scheduled for off hours. Anticipating that these boards will often have to share a line that's used for voice communications, the auto answer feature can usually be switched off, allowing you to activate the FAX board manually when it is needed.

A cut above the ordinary FAX modem, there is a class of FAX cards with their own on-board coprocessors, such as the Intel 80188. This completely frees them from the underlying system, allowing them to send and receive completely in the background while you continue to use your computer uninterrupted. This is multitasking in the truest sense of the word.

While bit-mapped FAX and traditional character-based data communications require quite different hardware and software, the hardware functions can be combined in a unit that still takes up only a single expansion slot, as in the case of the Hayes JTFAX 9600B that accepts a modem daughterboard, making it a complete communications package.

While all FAX, modem, or combination communications boards and their software can be run as background, operations in multitasking environments (such as DESQview, Windows, VM386, or others), it is not recommended except when using devices that have their own coprocessors. Otherwise, data can be lost while the main CPU's attention is diverted to attend to whatever other tasks are running. This is not to say that it cannot be done under certain circumstances, but there are too many variables.

Spoolers and buffered output

Another relatively cheap and easy way to slip around the 640K barrier—and one that can add greatly to productivity—is some form of print spooler. This really is a sort of poor man's multitasking, because it lets you keep right on working at your computer while software—or some combination of hardware and software—working in the background, keeps your printer busy churning out those reams and reams of paper that printers like to pile up on the floor.

In its most elemental form, a print spooler is simply a program—typically a TSR like MS-DOS's PRINT.COM—that, when activated, assumes control over a block of RAM it can use for temporary storage for output to a printer. In addition to DOS's rather primitive—though effective—spooler, there are some rather slick third-party spoolers on the market as well. Additionally, many word processors and other applications programs have some sort of internal spooling capability of their own.

Some spoolers—especially those that work from inside of an application—succeed in being almost totally invisible, not even adding any noticeable overhead. This is typical of those incorporated into some of the more powerful word processors like X-Write that let you keep on working uninterrupted—or nearly so at least—while printing.

There is a certain amount of overhead involved. PRINT.COM, for example, currently supports user-selectable buffer sizes from 512 to 16,384 bytes, in addition to the 5792 bytes required by the program itself. At most, this is rather modest by today's memory usage standards. However, to reduce overhead to an absolute minimum, some of the more sophisticated spoolers, like PrintQ and Printer Genius, use hard disk space rather than RAM for their primary storage space. In one case they use conventionally structured files that, if need be, can be called from DOS or stored in a format that only a spooler could love. They require some minimal amount of memory; however, by managing small blocks of data at a time, they can spool huge files while remaining unobtrusive.

Unlike DOS's PRINT and most of the give-away spoolers that come bundled with various expansion boards, better spoolers allow you to assign priorities, so first-in is not necessarily first-out. The order generally can be changed in midsession to accommodate sudden changes in priorities. Some even offer word processing-like formatting, editing, and other options not found in more typical dumb spoolers.

Another somewhat more expensive scheme adds special extra memory—a buffer—between your computer and your printer. While this is dedicated memory and cannot be used for anything else no matter what, it adds no overhead either. As far as your computer is concerned, it is completely invisible. You send a file to your printer and whatever your printer's internal buffer can swallow is held in the buffer and is released in small packets as the printer's internal buffer empties and makes room for more. (While internal buffer sizes vary greatly, printers typically have internal buffers of from 2K to 5K. A 2K buffer normally will hold the equivalent of about one double-spaced typewritten page.)

Such outboard buffers typically come in sizes from about 256K and up, with some capable of buffering several megabytes of queued files. Many come with minimum memory installed but are upgradable in increments, if and when a larger buffer is needed. The two options are not mutually exclusive either. An outboard buffer doesn't care whether it gets its input right from the original source file or after some middleman has come into the picture and possibly even modified the file in some way. However, it is unlikely that you would want to use both simultaneously.

One possible drawback to outboard buffering is that, in many cases, because there is no direct communication between the computer and the printer, you might not get the error messages you normally would get if you forget to turn the printer on, etc. Buffers are usually pretty dumb. They'll just hold whatever comes their way and hold it and hold it and hold it until someone discovers the problem, two hours after that very important and very long document was supposed to go out by special messenger. Still, the outboard printer buffer can be a very cost-effective way of increasing productivity. In many cases, it even adds the multiuser convenience of networking as well.

Peripheral sharing

Sometimes called peripheral sharing, the lowly buffer has grown up to include a whole class of devices, many of which are really little short of networking hubs with the general exception (though not necessarily total exclusion) of file sharing. With price tags sometimes ranging upwards of a thousand dollars for some of the better and more sophisticated devices, some of the peripheral sharing boxes allow the user to mix and match almost any mix of computers and printers up to the total number of I/O channels that it has. Typical of devices in this category are devices manufactured by Rose Electronics and by Buffalo Products. While the number of channels and configuration options vary widely, a unit with six channels typically can serve two computers sharing four printers (or three printers and a plotter) or two printers between four computers.

Not only can the same channels often be used either as inputs or outputs, but some also support two-way devices such as modems and many can be configured as either serial or parallel according to specific needs. Serial in and parallel out, or vice versa, is not uncommon, allowing great flexibility in configuring the overall system.

Whether serial or parallel, external buffers or peripheral sharing devices offer an added advantage in not taking up one of your precious expansion slots or adding more drain to your computer's power supply. In multiuser systems, they often are able to link even otherwise incompatible computers to one or more shared printers. Most also include internal buffers with sizes again ranging up to several megabytes but generally starting with some less amount installed. Priorities can be assigned, and the internal buffer can keep all printers on line running at capacity.

It all helps—but is it enough?

I've cut across only a narrow cross section of the options that are out there—relatively easy upgrades that will work regardless of the hardware platform you are using now. In many cases, something in this category might be all you need to meet your needs, perhaps just for now or perhaps for some time to come. Even when you do upgrade, you can and probably will take things like spoolers and external buffers with you. The peripheral sharing box I started with for one old IBM PC and two printers now connects three printers—still one old 8088 but now a pair of 386s, too.

Are any of these things enough, even for now? There is no simple answer to that question. If you still are working with an 8088 machine—or even something that is hotter but is configured pretty basically—they offer you a place to start at least. From there, the sky really is the limit.

12
CHAPTER

DESQview and the age of multitasking

A funny thing happened on the way to the 386: multitasking. It was real honest-to-goodness multitasking, with 8088s. This multitasking was not just the kind of swap-to-hard-disk task switching reinvented for DOS 5.0. It was not the feeble gesture IBM came up with in its ill-starred TopView only to be dug up again by Microsoft for Windows. (I hesitate to say resurrected because that implies some form of life.) You could multitask only whatever you could fit—in addition to Windows itself—on top of DOS inside 640K.

DESQview could do real multitasking. A lowly PC now could run code both in the foreground and the background. 286s could do it, too. They were faster, so multitasking was a little smoother. All you really needed, however, was an old PC, some extra memory, a program from a (then) upstart software company called Quarterdeck Office Systems, and a new kind of expanded memory: EEMS.

This is one of those fun stories where the technology guys get ahead of the pack and nobody quite knows what they've even got until somebody comes up with a way to use it. AST Research, a small creative bunch of hardware innovators best known for their memory expansion boards, had come up with a bank switching scheme. They called it EEMS (Enhanced Expanded Memory Specification), but no one really had a use for it until Quarterdeck picked up on it and saw the key that had eluded others. The rest, as they say, is history. Multitasking had arrived.

AST then pulled a clever marketing stunt. Because DESQview's multitasking magic only worked with AST's EEMS boards, the company started bundling DESQview software with its EEMS boards. The more you used DESQview, once you saw what it could do with EEMS memory, the more memory you had to have. More memory meant more EEMS boards.

You could buy DESQview separately and you could run it without the benefit of EEMS expanded memory. You could run DESQview with ordinary 3.2 EMS boards. DESQview, however, could not multitask with more than a total of 640K of programs (only what could be loaded at the same time into conventional memory) except when teamed with an AST EEMS board.

EEMS, if you'll recall from chapter 4, was a variation on the then accepted LIM 3.2 EMS specification. AST's EEMS conformed to that specification and did everything that EMS allowed. However, it added the ability to bank switch conventional memory, with that change, the industry was about to get turned on its proverbial ear.

Suddenly, bank switching up to four 16K packets of data in and out through a page frame up in the high address range was of only secondary importance. It was still okay for temporary data storage—for RAM disks, print spoolers, football-field-sized spreadsheets, and the like. By taking conventional memory away from the CPU and only loaning it back on a rotating on-demand basis, however, the whole world changed.

The heretics

EEMS was to some as heretical as when poor old Copernicus stated—and proved—that Earth was not the center of universe (and got excommunicated for his efforts). Someone had dared to say—and prove—that the CPU was not the center of the computer world, but rather just a servant in a bigger scheme of things.

The implications of this revolutionary concept were staggering, at least to those who chose to listen. There are, however, still those convinced the world is flat. From that point, you could almost divide the computer industry into two groups. There were the enlightened EEMS advocates in one camp. There also was something like a Flat Earth Society that surely boasted the designers of IBM's PS/2 models 50 and 60 among its charter members.

Intel, in the meantime, was caught on the horns of a dilemma. As co-authors of the LIM 3.2 specification and manufacturers of a respected line of memory expansion boards that did not embrace any of EEMS's more advanced features, Intel clearly had a serious problem. To admit that EEMS was better was a bitter pill to swallow. On the other hand, looking to its own new 80386, it had a chip that offered special multitasking capabilities that, at that point, could be exploited fully only with the kind of floating memory support EEMS afforded. Back at the ranch, some of Lotus's customers were unhappy with constraints in the old 3.2

EMS specification that as a practical matter limited expanded memory access to about half of the promised 8 Mb. (Lotus never even fully implemented the features of the 3.2 specification.)

Intel bit the bullet. The LIM alliance Gang of Three produced a new document, the LIM 4.0 EMS specification document. In name, it was still Lotus/Intel/Microsoft, but it was essentially AST's EEMS in a LIM cover. (According to inside sources, AST was not even consulted.) The new LIM 4.0 EMS did not include all of EEMS's features (although, interestingly, it does refer hardware manufacturers to and recommends compliance with one EEMS feature not included). However, it pretty much did—and does—include those features that had been proved so crucial to making multitasking a viable reality. Multitasking—and DESQview—truly had arrived.

To set the record straight, Quarterdeck did not invert multitasking, which had been around since the fairly early mainframe days. What DESQview did that turned the world around was to take advantage of the features that made AST's EEMS so special: the ability to bank switch conventional memory from 640K down to 256K (the 64K page frame called for in the LIM EMS specification was little more than just a bonus). With the ability to map 384K in and out of conventional memory though, things suddenly started falling into place.

Strangely, the old PC actually was better suited for multitasking than some of today's machines. At least with the old PC, you could set the DIP switches on the system board to show only 64K in the system (which was all the earliest system boards held anyway). Other memory then could be mapped into the addresses disabled by the switches. If supplied by—and at that time only by—an AST EEMS board, 384K of expanded memory could be mapped in and out of conventional memory addresses starting at 256K.

Given that, up to 384K of code and data could be plucked whole out of conventional memory address space and replaced by switching another whole 384K block in its place. This 384K is enough to run most DOS programs handily. By keeping programs and their data together, they could be kept alive and kicking—actually running (even in the background), not just out there in suspension, frozen-dead until they get called back again.

To do this, DESQview gave every program, whether it was running in the foreground or the background, some share of the CPU's time: time slicing. It was out of sight perhaps; however, no matter where it was, if your spreadsheet was recalculating when you swapped it out and brought another job up in the foreground it kept on crunching numbers. By the time you brought the spreadsheet back on screen again, the job likely was done. In the meantime, you did some other task. Almost any time-consuming task that did not require continued attention or keyboard input could be relegated to the background, which is especially valuable when you are running communications packages.

Admittedly, a 4.77 MHz PC, trying to divide itself among several different tasks, was something less than dazzling in the performance department. In a world where everything took longer than it does today, however, being able to run slow tasks unattended in the background meant even more than with the speeding demons on your desk today. Certainly, it started a revolution that quickly took hold in the marketplace. When the 80386 came on the market neatly packaged in a whole new generation of computers, DESQview was ready. With the 386 it became a faster, sleeker, and more powerful tool than ever.

A textbook 4.0 EMS case study

DESQview is really a case study in the use of EMS memory. An examination of that usage gives a good insight into its workings. Interestingly, the DESQview you buy today (packaged as DESQview 386 and bundled with QEMM) is the same exact DESQview sold as a stand-alone that could be run on any 8086-compatible machine. DESQview runs better on an 80386; however, its origins are rooted in the 8088/286 user base (before there was an 80386) and in the use of EEMS (now LIM 4.0 EMS), expanded memory before extended memory was a practical reality, so it really is based on a quite simple memory model.

For the moment ignoring the gyrations allowed by 386 and higher chips, DESQview loads—like any ordinary program—on top of DOS. Whatever conventional memory is left is available to use as a window in which to load and run an application. At this point, you haven't gained anything. On an 8088, you've actually lost the space DESQview needs for itself in conventional memory.

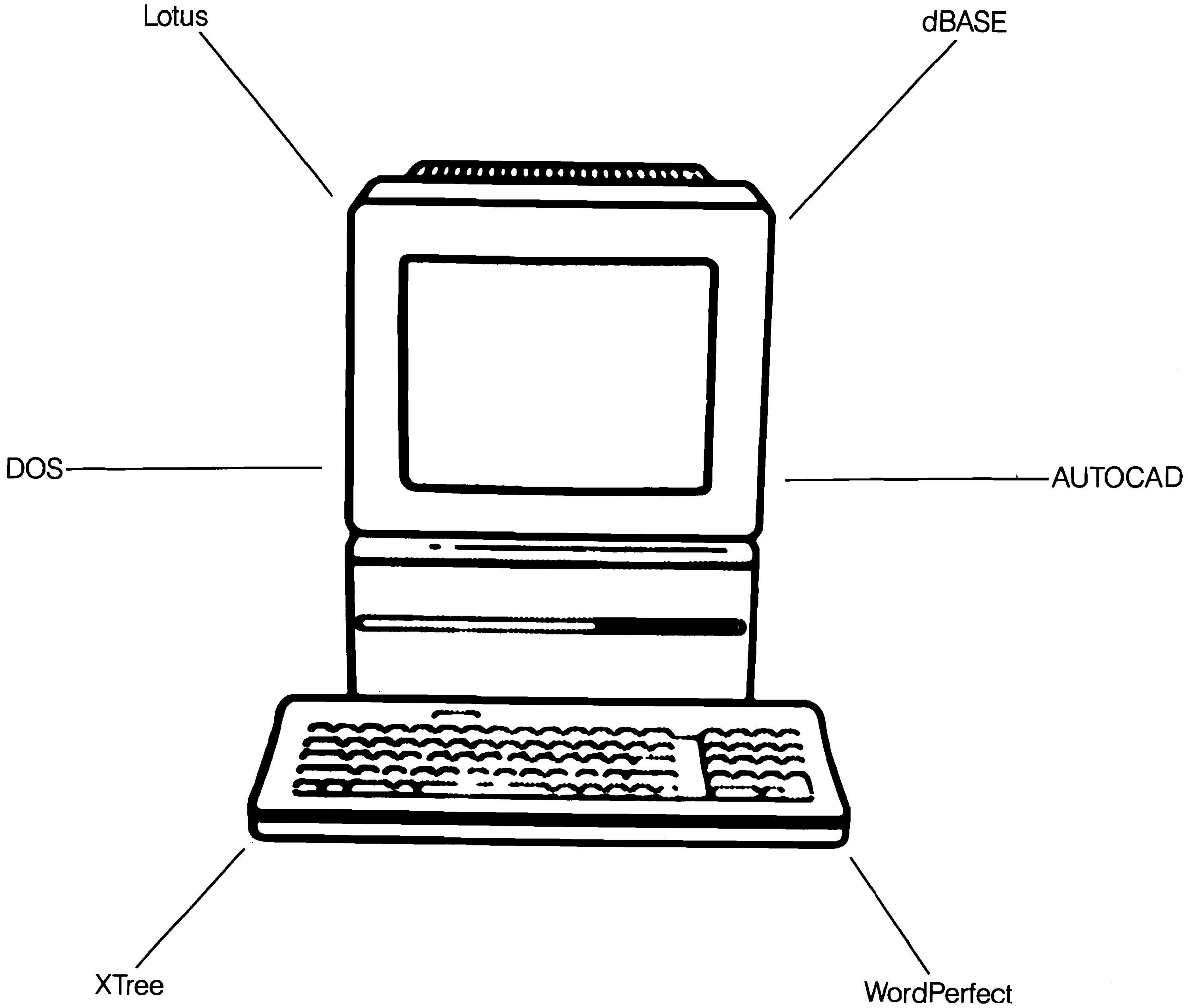
The payoff comes when DESQview is able to swap that application out into the EMS area somewhere—not just 16K pages squeezed out like toothpaste through the page frame to sit in limbo out there, but the entire block of running code and data (typically up to at least 384K, even in a worst-case situation on an 8088). Swap out that block and swap in another one—another window you already had another application running in or a fresh one that is ready to load, as in Fig. 12-1.

In this model, you are working with a single machine and a single DOS environment, which means that while you can change the environment available to an application within a window—the path, comspec, or any other variables—deleting some or adding others specific only to that window. Initially, the window will have inherited whatever environment was in place at the time DESQview was loaded.

The following lines illustrate the point. On the left, you see the environment before loading DESQview and, on the right, the environment as inherited by every window opened during that session:

```
COMSPEC=C:\COMMAND.COM      COMSPEC=C:\COMMAND.COM
TEMP=E:\                    TEMP=E:\
TODAY=03-01-1991            TODAY=03-01-1991
PATH=C:\DOS                 PATH=C:\DOS;C:\DOS\DV
PROMPT=$P$                  PROMPT=$P$
DIRCMD=/o:n/p               DIRCMD=/o:n/p
```

Both lists are identical, except that DESQview has insinuated the directory for its own files into the path. Like our own environment—the world we live in—we change the DOS environment a window has inherited. This solution is better than just packing the parent DOS environment with everything you possibly could want.



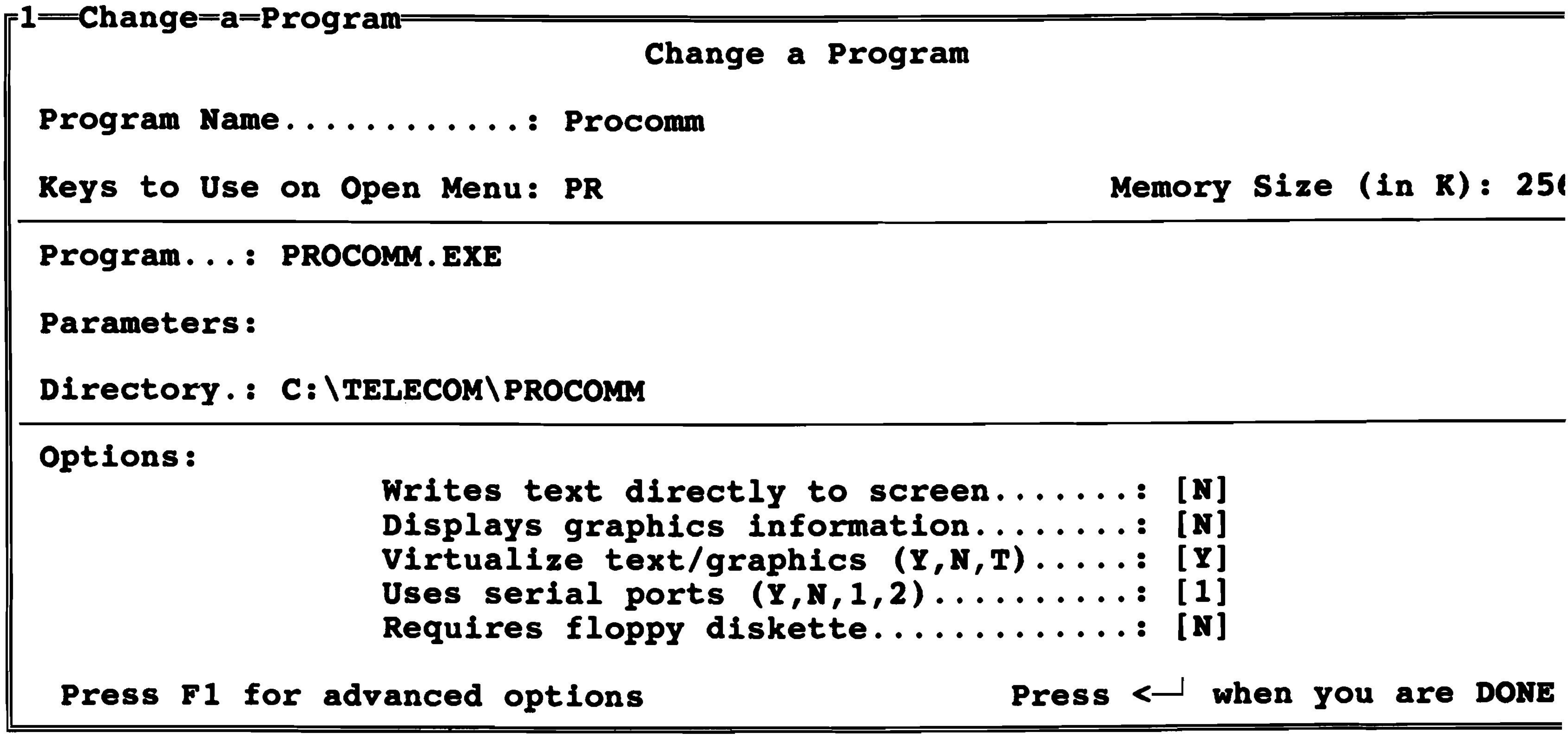
12-1 Reminiscent of the lazy Suzan analogy, as a practical matter, DESQview can load as many applications as you have memory for. However, it sits in the middle and, with 4.0 EMS support or when running DOS-extended programs, can access any running application almost instantaneously.

Stick to the straight and narrow

It is especially important that you stick to the straight and narrow when it comes to path statements, because the longer the path, the longer it takes DOS to get there (particularly when it comes to finding things out near the end of the line). Unless you specify exactly where a program is located when you enter the command (so DOS doesn't have to bother with the path), DOS always starts by searching through the default directory, then searches directory by directory through the path until it finds the right one—if there is a right one. DOS can't even tell you Bad command or filename until it has checked the whole list out.

One of the best ways around the problem is to not even set a path before loading DESQview. Programs given DESQview windows of their own do not need paths because that's taken care of when you create the startup file, as shown in Fig. 12-2. You'll note that the program is not called directly by DESQview in this case, but rather by a batch file. This setup often works out better because it allows you to set a path if need be—usually this is not required—or to make the directory where the data files are located the default directory and call the program from there, as in this case:

```
D:
CD \SUPRCALC\1991
C:\DOS\SUPRCALC\SC5
```



```
1══Change═a═Program══════════════════════════════════════════════════
                                  Change a Program

 Program Name............: Procomm

 Keys to Use on Open Menu: PR                         Memory Size (in K): 25
 ─────────────────────────────────────────────────────────────────────
 Program...: PROCOMM.EXE

 Parameters:

 Directory.: C:\TELECOM\PROCOMM
 ─────────────────────────────────────────────────────────────────────
 Options:
                Writes text directly to screen.......: [N]
                Displays graphics information........: [N]
                Virtualize text/graphics (Y,N,T).....: [Y]
                Uses serial ports (Y,N,1,2)..........: [1]
                Requires floppy diskette.............: [N]

  Press F1 for advanced options                  Press <─┘ when you are DONE
```

12-2 The DESQview CP (Change a Program) screen includes memory allocation and other often used—and need options. The advanced options screen allows broader range and allows specific settings.

This example does what the DOS APPEND command should do but doesn't. (The DOS APPEND command cannot even be used from within DESQview anyway, DOS will report a "Topview conflict" if you try to load APPEND.) This trick, however, does not work equally well with all programs. XyWrite, for example, must be loaded from the directory that contains a special startup file.

Realistically, about the only time you do want a DESQview window to have a path is for a blank DOS window—something that's handy for loading programs you don't use all that often, for experimenting with new programs, etc. Here's one that does all sorts of things automatically in the background every time the system starts, including some things that might ordinarily be in the AUTOEXEC.BAT:

```
@echo off
\dos\dv\dvansi >nul
c:|cd \dos
call addpath c:\dos\4dos;e:\  >nul
call \dos\batch\old_date
c:\dos\sitback\sb
mm >nul
Echo MAGIC MIRROR is Installed
Echo Scroll-C to capture, Scroll-T to transfer
prompt $P$_
set comspec=c:\dos\4dos\4dos.com
set 4dshell=/s:e:/u
call 4DOS
```

In the previous example, even the command interpreter and COMSPEC are changed to use 4DOS, rather than DOS's COMMAND.COM. This, however, applies only to the current window, as shown below in the environment on the left. The right side reflects the environment inherited by the next window to be opened, which goes on as if nothing has happened.

```
TMP=E: \                          COMSPEC=C: \COMMAND.COM
TODAY=03-06-1991                  TMP=E:\
PATH=C:\DOS;C:\DOS\UTILITY        TODAY=03-01-1991
PROMPT=$P$_                       PATH=C:\DOS;C:\DOS\DV
COMSPEC=C:\DOS\4DOS\
4DOS.COM                          PROMPT=$P$
4DSHEEL=/S:E:/U                   DIRCMD=/o:n/p
```

Sitback (c:\dos\sitback\sb in the above batch file) is a TSR backup program that runs in the background, automatically backing up any new or modified files that are located anywhere on one or more disks and meet specified criteria. It's a neat program, that is of special interest here because, when it is loaded in this manner (which is apparently the only way it can be loaded for use with DESQview, as attempts to load it before loading DESQview proved unsuccessful), it continually monitors hard disk activity, updating backups for files created or modified not just in this window but in all DESQview windows, on an ongoing basis. (There is also a window-specific version available for use with Windows 3.0.)

What makes Sitback especially interesting is the fact that when it is loaded this way, Sitback's roughly 16K of overhead is all charged against this window, giving other windows the benefit at no cost. This is not characteristic of the way most TSRs work with DESQview. Normally only those loaded ahead of DESQview (or any windowing environment) are available from within any window, so Sitback is a rare exception.

To build a better mousetrap

Because it was an innovative program, the developers of DESQview were faced from the beginning with a number of problems that, although not unique, were made more acute by trying to squeeze their multitasking environment into an already overcrowded 640K. DESQview is not a small program and the problem was serious—about 150K serious on an 8088. Necessity, they say, is the mother of all battles . . . or something. It was while searching for a way to reduce the impact of DESQview's rather portly overhead that Quarterdeck discovered DOS's extra 64K, now known as the High Memory Area (HMA). Because the HMA is most closely related to extended memory, Quarterdeck exploited its discovery by marketing QEXT, an extended memory manager for 80286s with HMA support.

At the time, DESQview was about the only program that could use the HMA—which was fine. With continued refinement, DESQview eventually was able to load no less than 63K into that 64K spot, significantly reducing its drain on conventional system resources.

The 80386, which brought with it the ready capability of mapping 4.0 EMS memory to unused address space between 640K and 1024K, offered intriguing new opportunities. Quarterdeck then developed a new memory manager for 386 and higher systems. It was an EMS driver with specific memory mapping support features that also incorporated the extended memory and HMA management features of its 286 driver.

With QEMM, Quarterdeck was able to develop a loading strategy that could spin bits and pieces of DESQview's code off in several directions, actually running sections of it in noncontiguous blocks of memory: 63K to the High Memory Area. QEMM could utilize the HMA more effectively than any other program to date. On the system I wrote much of this book on, some of QEMM's code was loaded in high DOS memory (including some mapped over nonessential areas in ROM), some out in expanded memory, and only a small residual—as little as 14K—down in conventional memory, as shown below:

```
First Meg / Programs
Memory Area   Size    Description
0D01 - 0E16   4.3K    COMMAND
0E17 - 0E1B   0.1K    COMMAND Data
0E1C - 0E2C   0.3K    COMMAND Environment
0E2D - 0E32   0.1K    COMMAND Data
0E33 - 0E3B   0.1K    XDV Environment
0E3C - 0E61   0.6K    XDV
0E62 - 11B1    13K    XDV Data
11B2 - 9FF1   522K    [Available]
= = = = =Conventional memory ends at 639K= = = = =
B000 - B0AA   2.7K    PCKRAMD
B0AB - B0AD     0K    SUPERPCK Environment
B0AE - B0AF     0K    XDV Data
B0B0 - B158   2.6K    LASTDRIV
B159 - B1CD   1.8K    FILES
B1CE - B60A    16K    SUPERPCK
B60B - B7FE   7.8K    XDV Data
CD00 - D271    21K    STACKER
D272 - DFFE    54K    XDV Data
F800 - F803   0.1K    XDV Data
F804 - F822   0.5K    MODE
F823 - FDFF    23K    XDV Data
      HMA      64K    DV
```

As you can see, this data, obtained with Manifest, not only shows how little conventional memory is being used by DESQview (XDV) but bits and pieces of its code and data are distributed, with over 23K (plus DOS's MODE.COM) relocated into memory mapped over recycled ROM addresses (F800h to FDFFh). Such specialized support can be obtained only by working with a memory manager written with a specific product in mind.

Negative overhead

The significance of this cooperative effort was such that, with the various tricks QEMM has at its disposal, DESQview 386 actually can save an amount of conventional memory greater than its own overhead. This means that in those circumstances DESQview 386 can create multiple multitasking windows, with any or all of them larger than would be available for single tasking without DESQview. (Assuming that no other memory manager capable of accessing the HMA and mapping memory to unused high DOS address space is installed.) This is better than stealing candy from a baby or getting something for nothing—nothing except a pool of memory large enough to support those windows.

Ultimately, the measure of performance still comes down to how much RAM you have installed. The more windows you want to have open at any given time, the more memory you need. That's pretty obvious, but what might not be quite so obvious is that the amount of memory required will generally be significantly less than 640K. The amount of memory necessary is typically in the neighborhood of 550K to 570K at most (DOS 5.0 with the kernel loaded in conventional memory to allow DESQview to have the HMA.)

The space required by DOS itself is a one-shot deal and a DESQview window can be only as big as the unused conventional memory that's left. This differs from VM386 and other multitaskers that use the 80386's virtual 8086 mode to create virtual machines. Every virtual machine requires its own copy of DOS and any other necessary overhead.

Still, effective multitasking takes a lot of memory. DESQview helps as much as it can, also providing virtual memory to swap inactive background applications to disk when the pool runs dry. There, however, is no substitute for the real stuff.

As discussed elsewhere in this book, there is a form of virtual memory usage (usually demand-paged) that allows the unused portions of the code to be swapped to disk, keeping those that actually are running in RAM. This is not the kind of swapping to disk DESQview does, if allowed, when it runs out of RAM. When DESQview swaps to disk, it's not selective; it just plain swaps. Whatever programs it swaps are stopped cold until recalled from disk to RAM again.

Demand-paged virtualization is something you're likely to run into only with very large programs of the type that can run in 32-bit protected mode through DOS extenders, such as Phar Lap's. The DOS extender, however, would have to perform the necessary memory management. Even if you had such a program, you would want to tell DESQview that it could not be swapped to disk because, as far as DESQview is concerned, it couldn't be unless you really mean to put it on ice.

To stretch whatever memory you have as far as you can, DESQview lets you set the amount of memory available to each window to whatever size is needed (it will round off to the next multiple of 16K). The real key to successful multitasking is careful management of whatever memory resources you have, using them as sparingly as possible. For example, except for major applications that need a lot of elbow room for data, when setting up a new window I generally try to pick a size too small to load the program. The window will simply abort and DESQview will return you to your previous window. The size then can be increased in 16K increments until it's big enough to fit.

Worlds in collision

For all its many benefits, the 80386 brought a whole new set of problems, too. DOS cannot run in protected mode. With the emergence of extended memory as a viable option with far greater potential than could ever be realized with EMS expanded memory, it was immediately apparent that there was need for a way for protected mode applications to run under DOS and use other real mode services (such as disk access, keyboard input, etc.).

Several developers worked on the problem. The result was a new kind of utility, DOS extenders, that can be linked to protected mode applications. Extenders could launch the programs from DOS to work in protected extended memory and return to DOS when necessary for such things as I/O services.

Whenever a DOS-extended application makes a DOS call or any other request that requires real mode, the DOS extender portion copies any necessary data down into the 1 Mb conventional memory area and switches to real mode. It then calls the requested function and switches back into protected mode, returning any results back to the protected mode operation.

To do this, the DOS extender has to have complete control over the system. This puts an extender in the category of something called a control program.

The problem is that protected mode normally allows only one control program to be active, which would limit you to running only a single protected mode program at any given time. To further complicate the situation, DESQview itself (or Windows or any multitasker) is also a control program, creating a situation in which it really wasn't possible to run a DOS-extended program in a multitasking environment. You could multitask or you could run the new, more powerful DOS-extended programs; you could not do both.

There basically was nothing inherently incompatible involved—nothing that could be resolved, at any rate. As has been the case on more than one occasion in this business, it was a matter of different players using different rules. It took some doing, but Quarterdeck got most of those involved at that time to sit down and try to sort things out as best they could, for everybody's benefit. Out of that emerged the Virtual Control Program Interface (VCPI).

There were those who weren't entirely happy with the VCPI specification. Windows, for example, was not compatible. In fairness, even aside from problems specific to Bill Gates' little darling, the VCPI does not address all of the issues, although those issues are such that a balance could have been struck and resulted in a single standard. For the first time, there was at least enough of a consensus to get the ball rolling—at least for those who had participated.

Windows' windows

DESQview has almost always supported multitasking for Windows-specific applications, such as Excel and Word, that were supplied with scaled-down runtime modules that allowed them to run graphically in nongraphical environments such as just plain DOS. Users who still have earlier versions of such programs still can, with the help of those runtime modules, run them in DESQview windows the way they have in the past.

Unfortunately, those runtime versions of Excel, Word and so forth will not even run on Windows 3.0. The new Windows is so different in so many different ways that only programs coded specifically for Windows 3.0 will run on Windows 3.0. As different as today's Windows is and as incompatible as it can be with older Windows software, DESQview 386 added special support features beginning with version 2.31 that allowed Windows 3.0 to run complete with application(s) from within DESQview.

As Microsoft kept patching bugs in the original 3.0 release, Quarterdeck kept refining its Windows 3.0 support, both as it affected not only running Windows 3.0 from inside DESQview but also direct QEMM support of advanced memory management features not provided by Microsoft's HIMEM.SYS (the version supplied with Windows 3.0 or even the more powerful versions supplied with DOS 5.0). I will discuss these issues in greater detail in the next chapter when I deal exclusively with Windows issues.

Alas, even with its speed, DESQview can't make Windows or Windows applications run any faster. Speed is just one of the benefits of the graphical user interface—any graphical user interface, not to throw rocks specifically in Microsoft's direction.

The tortoise and the hare

One of DESQview's strongest suits has always been its almost instantaneous response. To do all the cute things Windows—or any graphical interface—does, it has to operate in graphics mode, and that takes time. Every prompt screen and menu Windows shows you must be generated bit-by-bit in graphics mode.

DESQview's text-mode menus don't keep you waiting. Once you know, the commands things happen as fast as your fingers can find the keys. The only exceptions are loading programs into windows, which takes the same time it would take to load them anyway, and opening files. They can be done in the background while you're doing other things.

Outwardly somewhat similar to Windows in one respect, DESQview uses a form of PIF (*xx*-PIF.DVP) file to call up commonly used applications. Rather different in form than Microsoft's, they are also generally easier to create. For the newcomer, DESQview provides a set of startup files for a number of the more common applications programs. New *xx*-PIF.DVP files can be set up quickly using the easy, menu-prompted CP (Change Program) function. You can change existing call-up programs the same way.

As is typical throughout DESQview, even the CP function has two levels, depending on the user's expertise. The default CP screen contains only very basic information: program name, path, minimum space requirements, user-assigned two-character callup command, etc. Even some of the blanks come prefilled with default values suitable for many applications. A batch filename can be substituted for the actual program name when desired.

Earlier versions of DESQview had some problems with programs that write directly to the screen—as so many programs do. Such a program running in the background would not know that it didn't own the screen, so it would keep right on writing—right through whatever you had running in the foreground. This would be a nuisance, to say the very least. By version 2.0, however, Quarterdeck had solved that problem. With the proper settings in your *xx*-PIF.DVP files, the foreground program now owns the screen if you are using QEMM-386.

For power users and beyond

Another feature many users find convenient is DESQview's macro facility. Called "scripts," macros can be created via a learn feature that simply remembers and stores a series of keystrokes as you perform some operation and stores them to a key of your choice. These keystrokes are not stored in ASCII form. There is a utility with DESQview that can convert them to ASCII so you can edit them with your word processor. The same utility then converts them back into DESQview format so it can use them.

For power users with some programming experience, the DESQview manual documents several powerful customizing features. One of the most significant of these to achieve maximum multitasking efficiency is DV_PAUSE. If a program running in the background, for instance, is sitting idle with nothing to do, DV_PAUSE will relinquish the remainder of the program's time slice. This frees those otherwise wasted ticks to be used by any programs that might be running in the background. (This is different from the mechanism that DESQview uses to skip over a program that is waiting for a keystroke.)

For programs containing critical sections of code, there is DV_BEGIN_CRITICAL, which can be inserted ahead of a block of critical code so DESQview will not slice out of it. There is a corresponding DV_END_CRITICAL routine, as well.

Assembly language listings for the above and two other routines are included in the manual. For professional programmers, there's even a DESQview API (Application Program Interface). The API opens new horizons, so that today there are actually three types of applications that can be used in the DESQview multitasking environment:

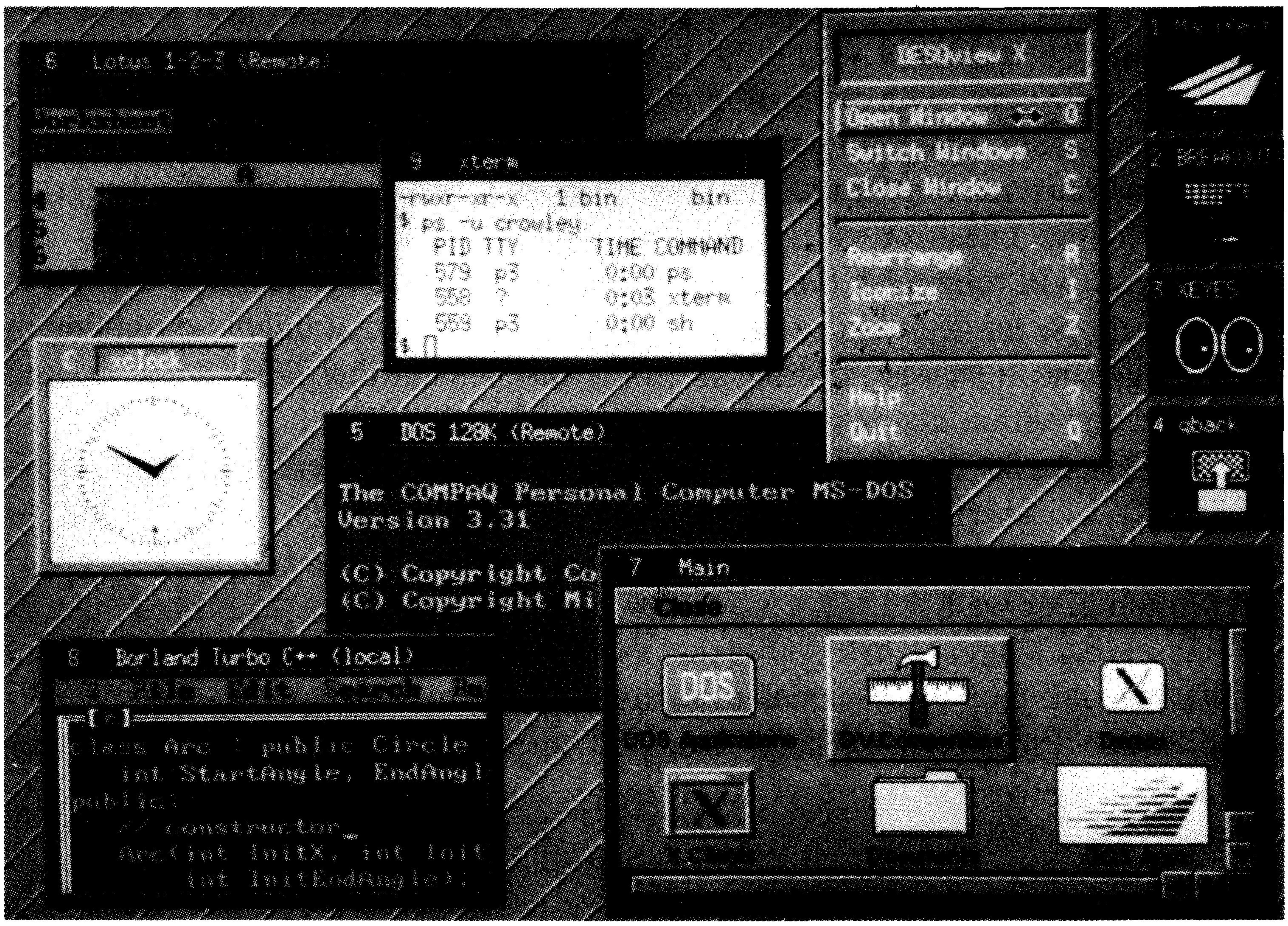
- DESQview-oblivious—these programs, including Lotus 1-2-3, Microsoft Word, and AutoCAD, know nothing about DESQview.

- DESQview-aware—these programs have been modified slightly by their developers to make them run more efficiently in DESQview. These include Paradox, dBASE, and WordPerfect.
- DESQview-specific—this is a much smaller group of programs that are written to take advantage of features found only in the DESQview API (Application Program Interface).

DESQview X

Now a third member of the DESQview family bridges the DOS/UNIX (or DOS/any operating system that supports X Windows) gap. The X Window System is a hardware- and operating system-independent standard that is designed to operate over a network or within a stand-alone machine, based on technology developed at MIT. Adopted as an industry standard by such companies as AT&T, DEC, Hewlett-Packard, IBM, Sun Microsystems, and others, it is the first commercial implementation of X Windows in the DOS environment.

The possibilities are awesome for DESQview X (Fig. 12-3), which was new at this writing. DESQview X allows PC users to participate in industry standard multivendor, multi-operating system distributed processing (i.e., cross platform


6 Lotus 1-2-3 (Remote)
Worksheet
9 xterm
DESQview X
Open Window O
Switch Windows S
Close Window C
Rearrange R
Iconize I
Zoom Z
Help ?
Quit Q
3 XEYES
4 qback
C xclock
5 DOS 128K (Remote)
7 Main
8 Borland Turbo C++ (local)
DOS
X

```
-rwxr-xr-x  1 bin       bin
$ ps -u crowley
  PID TTY       TIME COMMAND
  579 p3        0:00 ps
  558 ?         0:03 xterm
  559 p3        0:00 sh
$ 
```

```
The COMPAQ Personal Computer MS-DOS
Version 3.31

(C) Copyright Co
(C) Copyright Mi
```

```
class Arc : public Circle
   int StartAngle, EndAngl
public:
   // constructor
   Arc(int InitX, int Init
       int InitEndAngle);
```

12-3 Visually reminiscent of a Windows display, DESQview X bridges the gap between DOS and UNIX or other operating systems that support the X Windows protocols.

computing) from DOS-based machines. When DESQview X is networked, it will even allow 16-bit client machines to run sophisticated 32-bit applications on X-Windows servers, crossing hardware boundaries.

DESQview is not just one tool but a family of highly developed, sophisticated tools that are getting more powerful by the day. However, you don't have to be a pro or have a fancy 386 to join the club.

13

CHAPTER

Windows

If you believe many of the pundits, Windows is more than just the wave of the future; it is the future. Certainly, a review of the advertising in the periodicals would indicate a lot of vendors' futures hinge on catching that wave and riding it in. Certainly, Microsoft has gambled heavily on being a big enough fish in a small enough pond to generate a lot of waves.

At one time or another, Windows 3.0 has been on every hard disk in my house, but I think the wave rolled through here sometime back. There was a damp spot on the carpet one morning.

Seriously, however, Windows is a fact of life. Judging from the great disparity that there seems to be between the number of copies Microsoft has sold or given away and the number of applications that have been shipped, it has hardly taken the real world by storm. However, many people are using it. Certainly, there are a number of specific applications that cannot be run except in graphics mode—Computer-Assisted Design and Desktop Publishing for example.

However, graphics mode, by its very nature, imposes severe overhead on any system's resources. Until and unless video coprocessors come into general use, whatever benefits there might be will cost users dearly in overall performance. However, this is true of anything that's done in graphics mode using GEM or any other software platform. That, too, is a fact of life.

Windows 3, however, raises a number of specific issues, some that might not be immediately apparent to many users, especially with respect to memory utilization. Windows is a memory hog. In a way this is good, because, to breathe new life back into Windows, Microsoft was forced to crawl out of its 1 Mb shell even just to find room to run it satisfactorily. In doing that, Microsoft in effect, set the direction that the industry must go as DOS looks to its second decade: extended memory. Microsoft did not set that direction, rather the industry was already going that way. It had been since soon after the introduction of the 80386. What

Microsoft's de facto endorsement did was to legitimize that movement in a way no lesser god could have accomplished.

Unfortunately, Windows was incompatible with the existing VCPI DOS-extender specification. Microsoft had chosen not to be a party to its drafting. Rather than adapt to it, they then chose to try to force the acceptance of a significantly different specification of its own devising, the DOS Protected Mode Interface (DPMI). This move created a great deal of hard feelings in the industry.

The VCPI was not perfect by any means. At best a compromise hammered out by a number of developers all going off in different directions, it really represented the path of least resistance, requiring the least change by the greatest number of participants rather than an ideal situation. The DPMI, as it emerged initially, was hardly ideal either.

Ultimately, bowing to pressure, Microsoft reopened the matter and invited industry participation in revising the DPMI. The document that emerged from that differs greatly from the original specification. The resultant modified specification has implications reaching, seemingly, far beyond whatever the future holds in store for Windows.

For now, I need to discuss Windows and, in the context of this book, the way it impacts on the use of—and need for—memory.

About a ton of memory

Because Windows 3.0 requires such a prodigious amount of memory that it must look outside of DOS to adequately satisfy even its own needs, user awareness of what is required for effective memory management takes on a special importance. This is especially true in light of the fact that, by default, on installation, Windows pretty much just takes what it needs of your system's available resources without regard for any other needs.

This situation is made even worse by the fact that the memory management tools supplied with Windows 3.0—at least those supplied to date—are first order crude. They are so crude that, despite the fact that they bear the same names as the memory management utilities in MS-DOS 5.0, Microsoft specifically recommends that, if you have MS-DOS 5.0, you replace the Windows versions with their MS-DOS counterparts. In terms of overall system performance when running Windows, even those utilities lag far behind the recent memory manager releases from several third-party developers.

Fortunately, there are currently several good-to-excellent alternative solutions available to users of Windows 3.0. Quarterdeck, Qualitas, and All Computers, recognizing the problem immediately, incorporated special Windows support—something none of Microsoft's drivers have—into their memory managers.

Before jumping headlong into a discussion of the virtues of third-party management and how it actually can improve on what HIMEM.SYS and (on 386s or higher) EMM386.EXE—Microsoft's own managers—can do, we need to take a look at Windows 3s three modes and at the different problems they present—so different that all managers cannot address them all.

A three-Windows world

There are three Windows 3s. Each is so different that, aside from all being able to run Windows 3.*x*-specific software, they might as well be separate packages. Which one you run—which mode—depends entirely on the hardware that you have available; not just which CPU you've got, but what other resources you have, too.

Standard mode, despite its rather mundane name, is Windows at its best, provided you have enough RAM to support it. The documentation is a little deceptive, however, calling for only an 80286 with at least 1 Mb of memory (640K plus 256K of extended memory). In addition to providing access to extended memory (but not to virtual memory) this mode uses hard disk space in lieu of memory you don't really have. While this mode does allow you to switch among non-Windows applications, you need a bunch of memory to make that a practical reality.

The next mode is 386 Enhanced Mode. This one needs to be understood for what it is. It is a realization that probably most of the installed user base Microsoft would like to appeal to does not have enough RAM at their disposal to really support multitasking. Lacking that, the enhanced mode provides a back door: swapping to disk, or what is often euphemistically referred to as virtual memory. Applications written with this in mind can be designed to spin off chunks of code that are not needed immediately. To a point, this can cover for a lack of real live memory.

This says two things about enhanced mode operations:

- They will be slow. How slow they are depends upon the size of the applications being swapped back and forth, hard disk access times, and other factors. You can add two to five or more second of penalty time—most often more, I've found.
- Disk swapping is not multitasking, because codes cannot run, sorts cannot be performed, and numbers cannot be crunched while sitting on a disk.

Last—and least—there is Real mode. Real mode requires a minimum of Windows overhead, but provides a minimum of Windows functionality. This is Windows for the masses, Windows even for the 8088—provided that the 8088 has at least 6 Mb to 8 Mb of free hard disk space that Windows requires before it will even install.

You don't have to have an 8088 to run Windows 3.0 in Real mode, however. There are other reasons you might elect to do so. Not the least of these is that this mode provides the maximum compatibility with applications written to run under Windows 2.*x* (though still not enough to run some that I tried).

In any event, a closer look at both Real Mode and Standard Mode reveals that, to conserve whatever memory is available for Windows itself and Windows applications, Windows 3.0 always swaps some or all of a non-Windows application to disk, which, again, is not multitasking.

Discounting the Real Mode for which little can be done to enhance its capabilities, effective memory management is one of the most crucial factors in getting the most out of Windows 3.0. Here, Microsoft provides the Windows user with some basic tools; however, "basic" is the operative word.

What Windows 3.0 provides, and what it doesn't

The memory management device drivers furnished with Windows 3.0 (at least those seen to date), while having the same names as those now supplied with DOS 5.0 (HIMEM.SYS and EMM386.EXE), bear little resemblance to those namesakes. The Windows versions satisfy the bare minimum needs of Windows 3.0 and nothing else. Unfortunately, by stopping far short of the highly sophisticated memory management techniques demonstrated by any of a growing number of third-party memory managers and now even supplied with DOS 5.0, they would seem to do little to encourage anyone to use Windows 3.0.

Specifically, the Windows version of the EMS expanded memory emulator, EMM386.EXE, does not support the mapping of memory to addresses above 640K (referred to as UMBs in DOS 5.0) in a way that allows relocating device drivers and TSRs to those areas. While it is true that, even with such crude support, Windows 3.0 can use much of the available address space above 640K for itself, it also is true that that space can be used more effectively by more powerful memory management systems, like Quarterdeck's QEMM.SYS, 386MAX from Qualitas and All Computer's ALLCharge386.

EMM386.SYS is not a stand-alone however. It can emulate expanded memory only if it has extended memory to start with. No matter how many chips you have or how you have them wired, you need a device driver like HIMEM.SYS before you can have the extended memory to start with. Then, as the name implies, you only can use it on 386 (and higher) systems.

Although generally as lacking in frills as the bundled EMM386.EXE, the HIMEM.SYS supplied with Windows does support the High Memory Area, making it available for applications that can use it. It also offers a switch for HIMEM, `/hmamin=`*nn*, that can be included on the CONFIG.SYS command line to set a threshold level below which no application will be allowed access to the HMA. This is that peculiar extra 64K area that only one application can have the use of, so you need to be selective when using it. The typical syntax for these two drivers as they would normally appear in the CONFIG.SYS (assuming the two device drivers are in the root directory) is:

```
DEVICE=HIMEM.SYS /hmamin=40
```

```
DEVICE=EMM386.SYS 384
```

Here, I've specified 40K as the minimum size HIMEM will recognize as acceptable for HMA access. On the EMM386 line, I've specified 384K as the amount of extended memory to use in emulating expanded memory.

Other options supported by the Windows version of HIMEM.SYS include the number of handles available (the default is 32), whether to disable shadow RAM or not, and specifying the particular machine in use where special support might be required. Memory options supported by EMM386.EXE include Weitek coprocessor support, the establishing of a specific alternate page frame base address if a conflict occurs when the default is used, the exclusion of specific addresses from EMS page use, how many alternate register sets to emulate (the default is seven), and how many handles it can use (the default is 255).

While the Windows version of EMM386.EXE does not at this point support the loading of device drivers or TSRs into upper memory blocks, the EMM386.EXE supplied with MS-DOS 5.0 does. If it is loaded from the CONFIG.SYS, the AUTOEXEC.BAT, or the command line ahead of Windows, you can force the issue, making Windows do with what's left over. This still is not the ideal solution, however.

QEMM

According to some reports, Microsoft deliberately did not give users access to high DOS memory when it devised its memory mapping scheme for Windows 3.0, hoping everyone would standardize on Windows 3-specific software. That clearly didn't happen. Cost-conscious corporate America was not about to discard the hundreds of millions of dollars worth of applications software that was already in place and replace it with new software that not only was expensive to buy into but tremendously expensive to retrain their workforce for.

At the individual level, buyers played a lot of solitaire and waited—or tried to run their old nonWindows software and went back to playing solitaire. My non-Windows word processor, which loads and leaves me 337K for open files when running under plain old DOS, could barely load and gave me only 5K when running under Windows. My non-Windows spreadsheet would not even load.

Quarterdeck was quick to recognize the special needs of Windows 3.*x*—not only to recognize the needs but to devise workarounds that could be incorporated into QEMM beginning with its release 5.1. Meanwhile, others were working on the problem, too. The chronology, however, would seem to indicate that Quarterdeck was first to effectively crack the Windows code. Confronted with a *fait accompli*, Microsoft threw in the towel and offered to make its virtual device driver code available to Quarterdeck and other interested parties, so it could be incorporated by them directly into third-party memory managers, etc.

Giving away its actual V×D driver code like that presents an unusual situation; however, it had to have been clear to Microsoft that if Quarterdeck could do it, other people would be coming up with their own schemes—some close, some not so close. So the code was made available. Quarterdeck was apparently the first third-party vendor to actually market a package with that code incorporated in it.

For developers like Quarterdeck, there is another benefit beyond the obvious in all this, too. With Microsoft's cooperation at this level—having the actual Microsoft code to work with—the developers do not need to be concerned with making incremental upgrades as successive Windows 3.*x* versions come along.

Beginning with QEMM version 5.1, all succeeding QEMM versions (now in the 6s, to incorporate still further major changes occasioned by the release of MDS-DOS 5.0) have provided special support for Windows—Windows running under any of its three modes—on an 80386 or higher system.

The difference can be spectacular and has been reported as such by a number of power users. While Windows might have started out to be an area that Bill Gates claimed as his own to use and develop, it appears that Microsoft has cut itself off in a way that makes it almost mandatory to pass through third-party territory if you really want to get there.

QEMM replaces both HIMEM.SYS and EMM386.EXE, providing, as it always does, support for extended memory, the High Memory Area, Expanded memory, and memory mapping to upper memory blocks. However, you do have to specify how much of your available memory pool you want reserved for use as extended memory. This is the opposite of using MS-DOS 5s duo, which assumes you want extended memory unless you load EMM386.EXE and specify how much of it you want to use to emulate EMS expanded memory.

386MAX

386MAX (BlueMAX for PS/2s), certainly among the more powerful third-party memory managers, is compatible with Windows 3.0 in 386 Enhanced and Real modes only. Version 5.11 (and later) of 386MAX provides special support for Windows; however, this support is available only if specified during installation. With that support enabled, 386MAX requires more memory to run.

I'd like to point out something I discussed that might have slipped by unnoticed a little earlier: 386MAX itself does not support the DPMI, without which Windows cannot coexist with virtual mode programs. This is not a problem, however, because that support is intrinsic to Windows 3.0 itself and requires no specific support by the memory manager. 386MAX does support the more widely used VCPI specification, but this does not result in any conflict.

Among the more powerful Windows-specific features now included in 386MAX is a proprietary feature called Automatic Instancing. Without it, many programs that are not Windows-aware (TSRs and device drivers in particular) might not work the way you want them to in Windows.

ANSI.SYS is a clear cut case in point for instancing. Normally, it stores a record of the current screen colors where it is loaded into memory. That's fine until you want to use a different set of colors with a different window. Automatic Instancing gives you a copy of relevant drivers and TSRs in each of the multiple DOS sessions in Windows. This is not a problem unique to ANSI.SYS by any means either. There might be other cases where you might want different internal data accessed during different sessions.

The Windows documentation recommends you start your sessions first, then load your TSRs in each session accordingly. This doesn't always work, however. ANSI.SYS brings up a problem that Automatic Instancing addresses: normally, programs can be loaded into high memory only before Windows is loaded, because, given free rein, it will seek out unused address blocks above 640K and use them for itself. Yet, some of these programs need high memory access to function properly. It's a vicious circle.

There are a number of known TSRs and drivers that can benefit from Automatic Instancing. 386MAX has a list of these that it keeps internally, doing what the "automatic" in the name implies, so you don't have to worry about it. Obviously, 386MAX cannot anticipate every TSR and driver that needs this support, but it will catch a bunch of them. (Refer to the Windows 3.0 User's Guide for more information on the problem.)

386MAX also checks your system (when you boot) against known problems found in certain disk caches and SCSI disk controllers. There is a lot of smart built into this one, but then, when memory management is your only business, I guess you have to try a little harder.

386MAX also supports ROM shadowing under Windows. At boot time, it copies any ROM-based code—such as the VGA BIOS—into RAM for faster execution.

Everybody wants to get in on the act it seems. Windows 3.0 is able to take advantage of unused linear address space between 640K and 1 Mb to reduce its own memory overhead in low DOS. If Windows detects the presence of unused linear address space in high DOS at startup, it will map the addresses with physical RAM and load a small portion of its data into the remapped RAM. Because 386MAX might be mapping all the unused high DOS address space before Windows is loaded, you must be sure to leave some space unmapped to take advantage of this feature. Here again is one of those places a little trial and error is required to get the most out of your system.

If the 386MAX installation program detects an available monochrome display area, it will map only the first 8K of the available space by inserting appropriate USE= statements in the options profile. By doing so, Windows will take advantage of a portion of the remaining space (the actual amount used will vary).

If 386MAX.SYS has detected a nonsupportive busmastering controller and SMARTDrive is not loaded, then an error message will result when loading Windows 3.0. If you are sure you have a VDS-compatible busmastering controller, place SET BUSMASTER=VDS in your AUTOEXEC.BAT to bypass this error message and load Windows 3.0. It should be noted though that using these options without adequate knowledge of your system setup can result in the loss of data, so look before you leap on this one.

Although Windows will make use of a portion of the unmapped space, it might not always be the most effective use of the space on all systems. This is a result of the fact that the unmapped space could have been mapped as high DOS by 386MAX and used to store resident software, which can be forced to reside in conventional memory. With the help of a couple of 386MAX's utilities, MAPMEM and MAXIMIZE, a little trial and error here can result in better usage of this space. The procedure is well documented.

It should be noted that Windows 386 (as opposed to Windows 3.0) is not compatible with 386MAX. That earlier version does not support a standard interface that allows virtual mode programs to coexist successfully. However, you use the WIN86 portion of WIN386 by installing according to the instructions below for WIN286. Also, Windows 286 is fully compatible with 386MAX. For proper support of all Windows 286 features with 386MAX, you must use a copy of Windows dated July 1988 or later (the date of your version can be found by listing the directory of your Windows SETUP diskette).

ALL CHARGE 386

All Computers has built special Windows 3.0 enhanced mode support into its ALL CHARGE 386 memory management package beginning with version 3.1. This release also incorporates All's automatic optimization program. There is no need to reconfigure manually. You simply have to run the installation program and ALL CHARGE 386 will update your existing configuration automatically.

Aside from providing memory management that's far superior to Microsoft's HIMEM.SYS and company, of greatest interest specifically to Windows users is the fact that this updated version provides the instancing support that is so vital to the successful use of many TSRs, ANSI.SYS, and other similar device drivers in the Windows 3 environment.

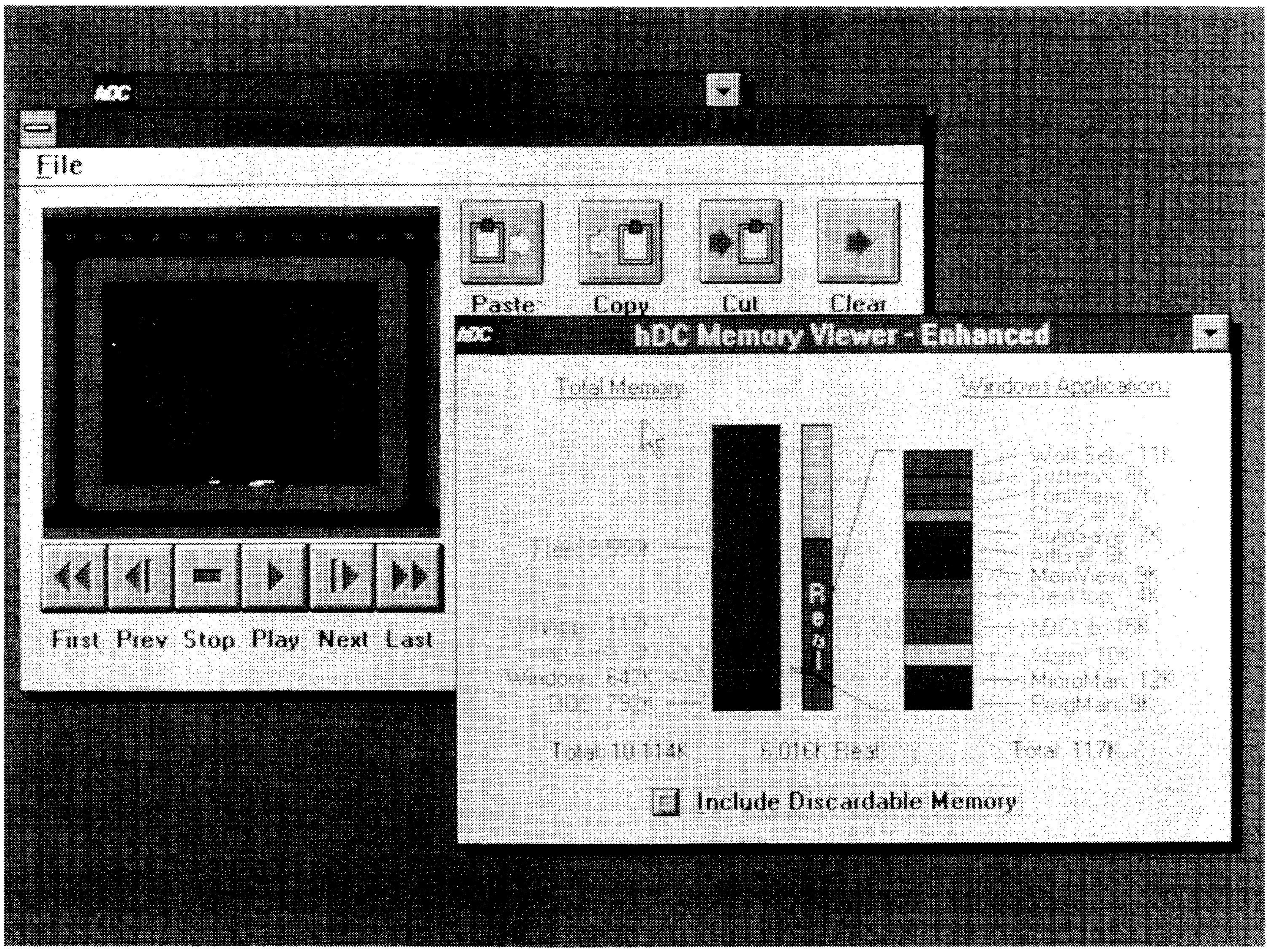
Registered users having earlier releases can upgrade at a cost of only $5.00 to cover shipping and handling. For information call (800) 627-4825.

hDC FirstApps

The difficulty in seeing just what you are doing, one of the greatest obstacles to effective memory management, seems even more frustrating under Windows. Working with 386 or better systems, any of the top third-party memory managers provide utilities that enable you to see how effectively you are using mapped memory. They can be run under Windows in the same way as any non-Windows application.

There is also another utility package that Windows 3.0 users might want to consider: hDC FirstApps. Actually a package of utilities, FirstApps has a Memory Viewer function that shows what's where, regardless of what hardware platform or which mode Windows is running under.

Memory Viewer opens a window that, as shown in Fig. 13-1, graphically displays all of the available memory on your system and how it is being currently utilized. The data displayed depends on whether you are running under Standard, Enhanced, or Real Mode.


File
Paste
Copy
Cut
Clear
First
Prev
Stop
Play
Next
Last
hDC Memory Viewer - Enhanced
Total Memory
Windows Applications
Real
MemView: 9K
Desktop: 14K
MicroMan: 12K
ProgMan: 9K
Total 10,114K
6,016K Real
Total 117K
Include Discardable Memory

13-1 FirstApps from hDC provides helpful information about Windows memory usage as part of utility package.

Starting at the bottom in Real Mode, Memory Viewer displays one or two barcharts (depending on whether or not you have expanded memory showing how all of your available memory is allocated). In Standard Mode, the Memory Viewer also displays two bar charts. The one to the left shows how all the memory on your system is being used; the other shows how memory is being used by currently running Windows applications. Enhanced Mode adds a third bar chart, showing how current memory usage is divided between real memory (RAM) and virtual memory (i.e., code and data swapped to disk, which therefore is not actually running).

The Memory Viewer also has one additional option that is interesting: include Disposable Memory. This displays the minimum amount of memory an application can run in. A good part of this information is of interest specifically to Windows users and, as such, is not readily available from other sources.

Quite aside from the debate over Windows 3 and whether users really need or even want a Graphic User Interface or whether the DPMI will really play a role in shaping the direction of the future of the industry, there are a number of more immediate issues users must address when thinking about Windows. Among those, having lots of memory and managing that memory effectively have got to rank among the most important.

14
CHAPTER

Beyond the real: the virtual machine

Certainly one of the most unique features of the 80386, the *i*486 and surely anything else that comes down the pike at this point is the ability to support not only real mode machines (the only kind most users are familiar with), but also almost any number of virtual machines, all running concurrently. Such virtual machines can be used for multitasking by a single user on a single physical machine, several different users at remote locations, or a combination of the two.

The virtual machine is probably the least understood of all the capabilities of these higher CPU chips. They are exotic both in concept and in execution. They really are essentially just an extension of the protected mode and perhaps demonstrate the real meaning of protected mode more graphically than any ordinary usage.

Virtual mode is more correctly called virtual 8086 mode because that is what the CPU is emulating when it runs in virtual mode—multiple 8086 processors. The illusion of being not one but multiple processors is so complete, as implemented in these chips, that multiple operating systems can run concurrently under a supervisory device. A 386 machine, for example, might boot up in real mode under DOS but yet play host to virtual machines, one or more running UNIX tasks while still others might be running under DOS or even a mix of different versions of DOS or something else.

It's no illusion. You actually have to boot your virtual machines the same as you would boot a real machine. Booting under DOS, you need a CONFIG and AUTOEXEC file for each. If it would take 30 seconds to boot up your real mode machine to start the day, and if you want to run three virtual machines, you've got to start by booting up a real machine and then three more machines. That's a couple of minutes anyway, not including loading your applications software on each one. For that reason, it is significantly slower getting up and running with a bunch of virtual machines than DESQview or probably even windows in this one respect.

You must remember, though, that no matter how real the illusion of the virtual machine might be, you still only have one CPU that must be shared among however many different "machines" you might be running, as in any multitasking situation. Even aside from that, however, as implemented in VM386, once you're booted up and running on your virtual machines, they typically will run a little slower than real machines in actual operations too. The difference is just enough that you might notice, but not enough to really slow you down in most cases. There is a tradeoff involved between performance and protection. Even at that, virtual machines will still run rings around Windows 3.0 in any mode, unless you're running Windows on a virtual machine for any reason.

If it will run on any 8086 machine . . .

The important thing to keep in mind is that, if a program can be run on any real 8086-family machine, it probably will run on a virtual machine. That seems to be, if not the sole, at least the primary criterion. There are a few notable exceptions, including the DOS FORMAT command, but I'll discuss some of these a little later.

Each virtual machine can have its own extended and expanded memory. The difference here is that, because virtual machines run only in protected mode, their memory (once allocated during the bootup process) is the exclusive property of that machine for the session. This is despite the fact that up until that point it was—and except for the virtual machine would remain—part of a common pool. When you terminate a virtual machine (short of shutting off the host machine), that memory, however, is released.

There is one important difference that must be noted on the subject of memory management. Programs, like VM386, that create and manage virtual machines do their own memory management. They will not even work in the presence of another memory manager, like QEMM or DOS 5's HIMEM.SYS. Whatever DOS you use must boot up pretty bare (typically FILES and BUFFERS are the only items in the CONFIG.SYS you'd use to boot your host machine). The CONFIG and AUTOEXEC you use to boot up individual virtual machines look more like what you would expect.

Theoretically, the ability of an 80386 or higher chip to create virtual machines is almost unlimited. The entire 4 gigabyte address range is available for remapping. As a practical matter, however, it is not as unlimited as it might seem at first glance.

No virtual clock

No matter how clever the illusion is, you still come back to time slicing. There are no virtual clocks, just one physical clock.

With memory as cheap as it is these days, the clock becomes the ultimate limiting factor in the equation. Fortunately, developers have devised some clever work-arounds to squeeze the most out of that finite resource: sensing periods of inactivity and devoting more than just the allocated share to CPU-intensive tasks on other virtual machines.

In a single user multitasking situation, you usually can stretch your precious time slices, using them only where they'll do the most good, by simply freezing any background operations that do not have to keep on processing. Fully half the applications I use in a normal day fall in that category.

On a multiuser system, however, things are different. Jobs that appear as background activity from the perspective of someone working at the box that houses both the host as well as his or her virtual machines are the foreground tasks for other users on the system and are, to each of them at least, of equal importance. You would like to keep all those other users working, not just sitting there because, the moment they have to sit and wait their turn, a multiuser system is no longer viable.

In addition to sharing a common internal clock (which is something any multitasking/multiuser system has to do), designing systems around virtual machines raises other problems that are unique. All users must share a common I/O interface that in the case of multiuser systems must, with the exception of shared resources such as disk drives, deal with different physical devices. Even shared devices must have special interfacing.

Whatever problems the virtual 8086 might pose, developers have found solutions. Now powerful and unique new systems are appearing on the market.

VM386: multitasking and more

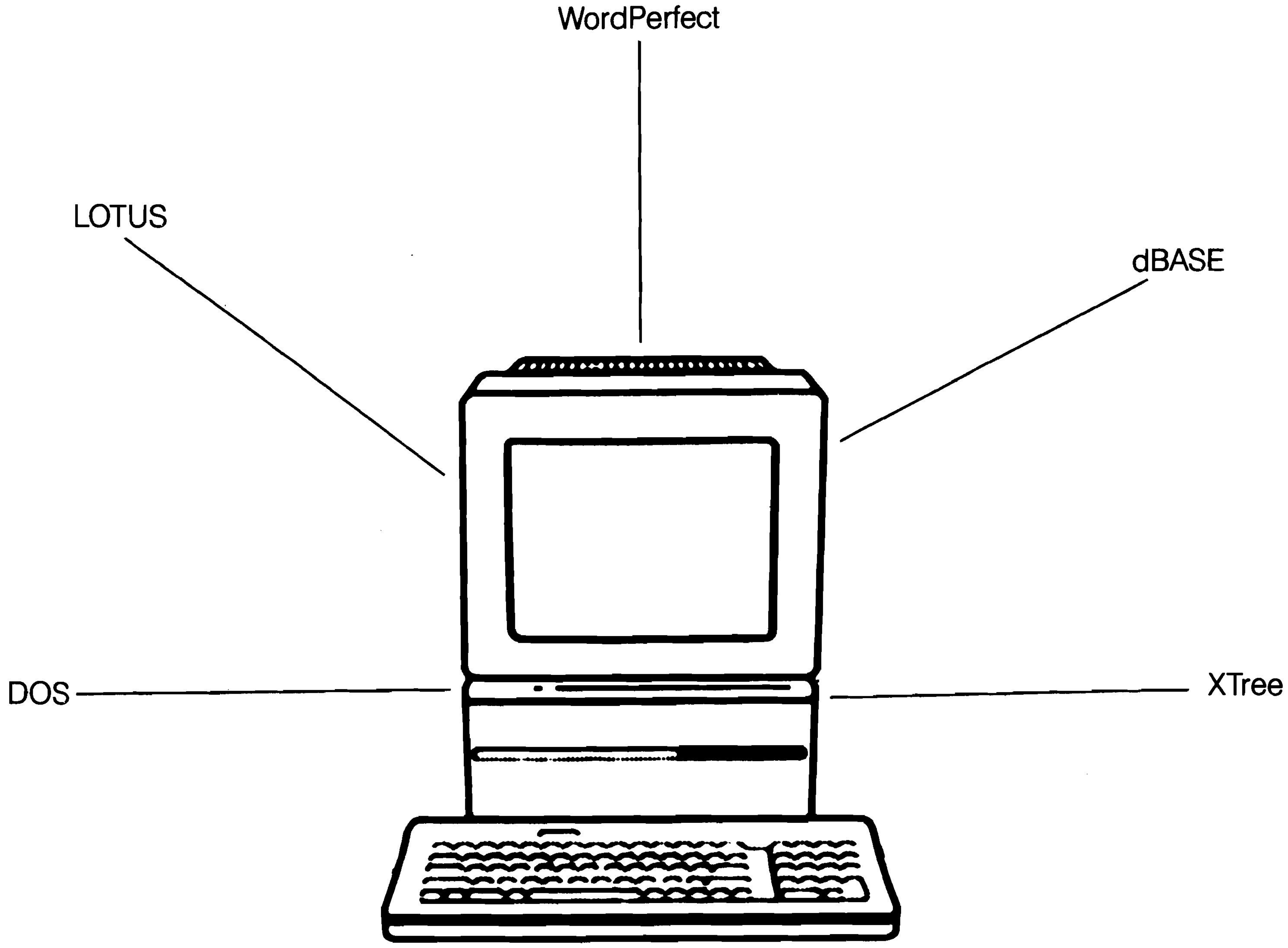
Although VM386 from Intellegent Graphics Corp is only one of several packages written exclusively for virtual 8086 mode operations, it was one of the first to attract much notice and is still one of the most interesting packages. I picked it for inclusion in this book for several reasons. Not the least of these reasons is, in my opinion, the fact that, from the basic entry-level single-user multitasking platform, VM386 can be developed and expanded in progressive orderly steps to build increasingly complex multiuser systems that cannot only share such resources as modems and printers but also be linked to networks.

Like DESQview, Windows, and other traditional multitaskers, the IGC VM386 packages are perfect supersets of DOS—they run on top of MS-DOS. There is a definite advantage to going this route, rather than striking off on its own to develop a whole new DOS look-alike environment as some developers have done with varying degrees of success. Some of the look-alikes have come a long way, but compatibility problems do emerge from time to time.

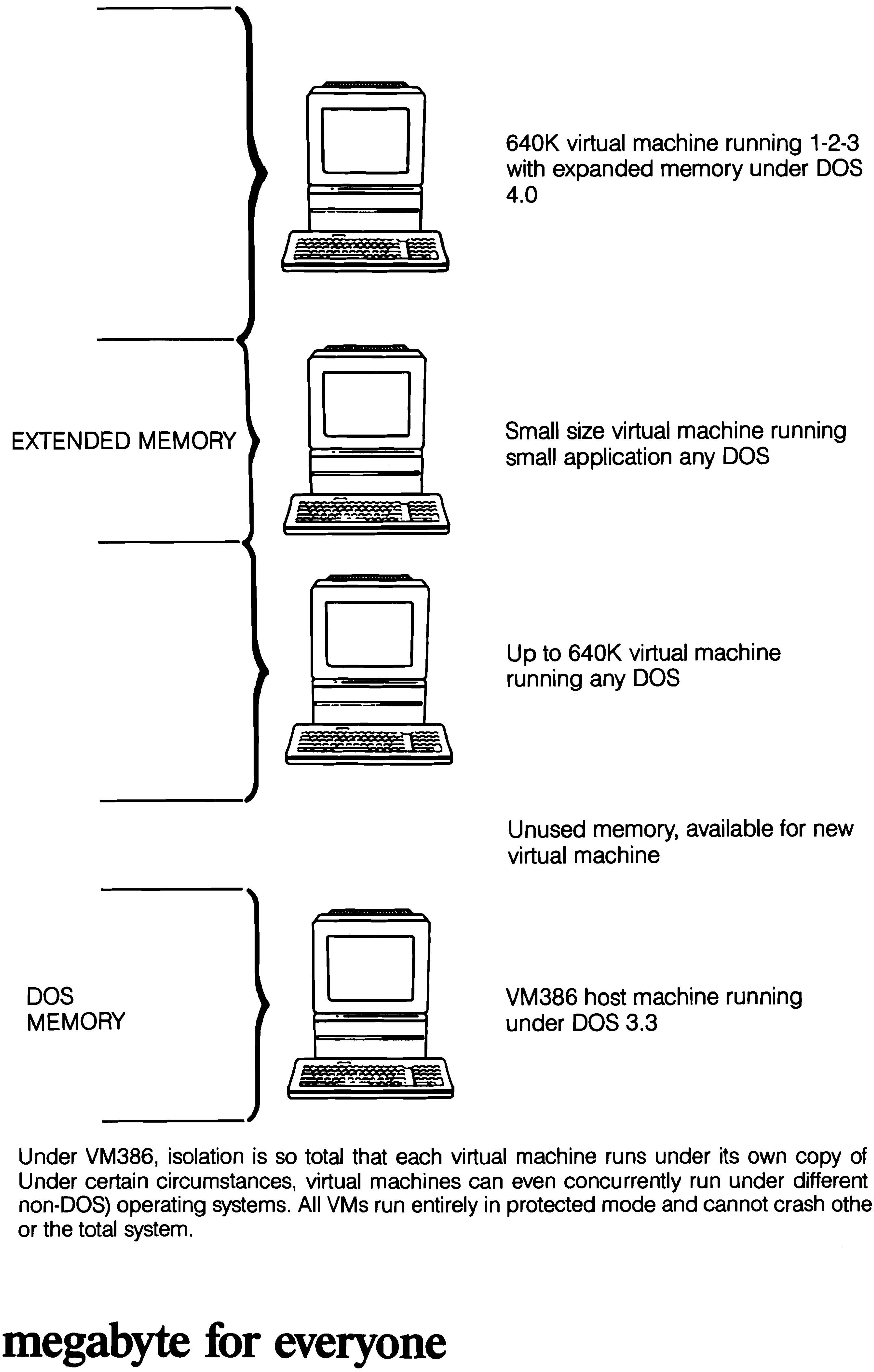
Sharing a common DOS platform is where similarities end. Unlike DESQview or Windows (which do their work within the confines of a single machine), VM386 creates separate machines. These virtual machines all run in protected mode, completely isolated from each other and from the real mode machine that created them.

To those unfamiliar with virtual machines and the virtual 8086 mode, this concept no doubt sounds more like a matter of semantics and a bunch of advertising hype. It, however, is more than that.

With more familiar multitaskers, you really are always working within the confines of a common single DOS environment on a single physical machine, as shown in Fig. 14-1. In that environment, if one of your applications hiccups, it will sometimes bring your entire system down, with the possible loss of data in whatever other windows you have open at the time. By contrast, the virtual machines of VM386 (Fig. 14-2) might as well be across town from each other because they are completely separate.



14-1 With tradition multitasking, even on a 386 or 486, all tasks share a common DOS, inheriting a common environment. Applications simply are switched in and out of the foreground.



640K virtual machine running 1-2-3 with expanded memory under DOS 4.0
EXTENDED MEMORY
Small size virtual machine running small application any DOS
Up to 640K virtual machine running any DOS
Unused memory, available for new virtual machine
DOS MEMORY
VM386 host machine running under DOS 3.3

14-2 Under VM386, isolation is so total that each virtual machine runs under its own copy of DOS. Under certain circumstances, virtual machines can even concurrently run under different (even non-DOS) operating systems. All VMs run entirely in protected mode and cannot crash other VMs or the total system.

A megabyte for everyone

Because VM386 virtual machines conform to all the ordinary rules of DOS, each virtual machine has one full megabyte of address space starting down at 0000h and running up through FFFFh. This space is apportioned in the usual way: 640K for programs ad the operating system, with the rest reserved for system use (video, etc.). You can even have an EMS page frame tucked in between the video and ROM regions, if you like, just like a real machine.

How can all of these virtual machines use the same address space at the same time? If you look carefully at Fig. 14-2, they only seem to be using the same address space. This is one of the features unique to the 80386 and *i*486 chips—the ability to lock onto a contiguous chunk of protected memory anywhere in the up to 4 gigabytes that the chips can address and present it as if it really started down at 0000h. The addresses, as seen by DOS and by your applications, appear like normal conventional memory addresses up to 640K and like whatever else they're supposed to look like after that.

Outside of the usual megabyte, VM386 supports both extended and expanded memory. The amount of each—if any—that will be available to any virtual machine is determined at the time that machine is booted by parameters established when the machine is created. This, in turn, determines the size of the block of contiguous memory that is set aside for the exclusive use by that machine. Once allocated, that memory belongs exclusively to that machine for as long as that virtual machine exists—the duration of that work session or until it is specifically terminated—whether it actually is used or not. There is no sharing.

Although VM386 does support both extended and EMS expanded memory, it does not currently support the High Memory Area or mapping EMS memory to high DOS addresses. This limitation, however, is not imposed by the virtual 8086 mode, but rather these are features that simply have not been implemented at this time. As a practical matter, the fact that each application runs on a machine specifically tailored to its needs rather than having to load device drivers and TSRs in anticipation of other needs, and the fact that machines running text-only applications can include the memory area above A000h normally reserved for graphics, significantly reduces the need for such support.

With real or virtual machines, not all applications need the full 640K that DOS allows (or 704K or more for text-only applications). Let's face it; a lot of us survived rather well for quite a while on 256K or less at one time although it would be impossible with much of today's software. Still, with only a couple of exceptions, I find most of the applications run quite nicely in a scaled-down workspace.

Just like any machine, you have to allow space for DOS. In this respect, multitasking with virtual machines requires more memory than ordinary multitasking, because each virtual machine must have its own individual copy of DOS rather than sharing a single common DOS environment. Even at that, in many cases, you can run an application on a smaller machine. To conserve memory, VM386 allows virtual machines to be created in almost any size. (The normal increments are 128K, but advanced users can work with smaller increments, if desired.)

The configurations that determine machine size and memory allocation when a virtual machine is created can be saved and used again or used for just one session as desired. Where configuration files are saved, system startup can be automated, including the booting of several machines and loading of specific applications on them.

It is important to note, however, that once allocated when you boot a virtual machine, whatever extended or expanded memory you might have set aside for that virtual machine belongs to that VM exclusively. On the down side, that means that, even if that memory is not used and another VM needs it, it can't have it—at least not as long as the machine that has it is up and running (virtual machines can be terminated in mid-session and, at that time, whatever memory they were holding is made available again).

With a little management on your part, this shouldn't be a problem. It is a small price to pay for the protection that it buys. It is the way protected mode—the only mode these virtual machines can run in—guarantees protection for the programs and their data. Memory is not just one big pool for everyone to draw on as users are used to having it.

A clean environment

On an ordinary machine, the environment is something that is created initially on the basis of certain entries in the CONFIG.SYS and AUTOEXEC.BAT. People often talk about the DOS environment in the sense of the whole DOS work area; however, within that, buried somewhere in the bowels of memory, there is a tiny area that is technically called the environment, as shown in this excerpt from a Manifest display of first megabyte memory usage:

```
              First Meg / Programs
Memory Area   Size    Description
0D01 - 0E16   4.3K    COMMAND
0E17 - 0E1B   0.1K    COMMAND Data
0E1C - 0E2C   0.3K    COMMAND Environment
0E2D - 0E32   0.1K    COMMAND Data
```

There it is: 0.3K. Actually it is just 256 bytes, and less than that with DOS releases previous to 5.0. The following is an ASCII dump of that area done with System Sleuth.

```
OFFSET-->   0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
PARAGRAPH
0E1C:0000   M                       ≤   ñ   X   Y   –       Φ   ó
0E1C:0010   C   O   M   S   P   E   C   =   C   :   \   C   O   M   M   A
0E1C:0020   N   D   .   C   O   M       T   E   M   P   =   e   :   \
0E1C:0030   T M P = e : \   T O D A Y = 0 3
0E1C:0040   – 0 5 – 1 9 9 1   P A T H = C :
0E1C:0050   \ D O S ; \ C : \ D O S \ B A T
0E1C:0060   C H ; C : \ D O S \ U T I L I T
0E1C:0070   Y   P R O M P T = $ P $ –     =
```

There's more, but this is enough to see that this is the source of the environment DOS displays at the SET command. Note the new DIRCMD parameter that customizes the way DOS displays directory listings (this one sorts them alphabetically, displaying only one screenful at a time):

```
C>SET
COMSPEC=C:\COMMAND.COM
TODAY=01-13-1991
PATH=C:\DOS;\C:\DOS\BATCH;C:\DOS\UTILITY
PROMPT=$P$_
DIRCMD=/o:n/p
```

This is probably a fairly typical DOS 5.0 environment. As you can see this is where DOS stores important information like the path and other things that users think are important. The only one that DOS really cares about is the COMSPEC, which tells DOS where to look to find the command interpreter when it needs it. Without a COMSPEC—or if no valid command interpreter is found there—you've got a problem.

Working with a windowing-type multitasker, the windows each inherit a copy of the original environment from DOS. Subsequently, every window you open inherits its own copy of this original DOS environment: the path, the prompt, the comspec—everything. This is a fundamental rule of DOS and DOS machines. A child process—any child process—inherits the environment of its parent. The following lines are the environment one of those children, a DESQview window, has inherited—a stepchild actually:

```
C>SET
COMSPEC=C:\COMMAND.COM
TODAY=01-13-1991
PATH=C:\DOS;\C:\DOS\BATCH;C:\DOS\UTILITY;C:\DOS\DV
PROMPT=$P$_
DIRCMD=/o:n/p
```

With the exception of this window's most immediate parent (DESQview) having insinuated the location of its files into the path, this is the same as the original DOS environment before anything was loaded.

Now, by contrast, see what a VM386 virtual machine gets when it boots. It doesn't matter here what all was in the original environment; this is what you get:

```
C>SET
COMSPEC=C:\COMMAND.COM
```

DOS always points to the copy of the command interpreter associated with the disk it was booted from and, by default, establishes that copy as the one that DOS looks to whenever it needs a command interpreter. DOS automatically creates a COMSPEC = statement in the environment on that basis.

So where did the rest of the environment go? It didn't go anywhere. It was never there. That's the point. This is a completely different machine, even if it does reside in the same box. If you want something special in the environment—a path, a prompt, a different COMSPEC perhaps—you have to put it there.

Split personality

One of the factors that contributes to reducing overhead on individual virtual machines is the fact that only the device drivers and TSRs required by the applications that will run on that specific VM need be loaded. Each virtual machine really is a custom machine in every respect. With this in mind, virtual machines can be given the names of the specific applications that they will run, with matching customized sets of CONFIG and AUTOEXEC files for each. You really have to have customized CONFIG and AUTOEXEC files if you want your new machine to come up with anything but a blank screen prompting you to input the time and date. This is where you really start to see and feel the difference.

For instance, I created CONFIG.SC and AUTOEXEC.SC for a Supercalc VM. Those are the files VM386 reads and executes whenever it boots the Supercalc virtual machine. By the same token, XyWrite has its own virtual machine and custom startup files. Here is the startup CONFIG file, CONFIG.XY, which looks exactly like the CONFIG.SYS for a real mode machine, and excerpts from the AUTOEXEC.XY file, showing several startup commands unique to VM386 virtual machines:

CONFIG.XY

```
DEVICE=C:\DOS\VM386\
  VMVDISK.SYS
DEVICE=C:\DOS\VM386\
  VMEMM.SYS
LASTDRIVE=Z

FILES=20
BUFFERS=15
```

AUTOEXEC.XY

```
C:\DOS\VM386\
  VMLINK.COM LPT1 * FL
C:\DOS\VM386\
  VMFSS.COM
C:\DOS\VM386\
  VMID.COM
C:\DOS\VM386\VMFG
CD \DOS\XYWRITE
EDITOR
```

Because, for all intents and purposes, you are working with separate machines when running under VM386, little things like sharing access to such system resources as printer ports and disk drives are not automatic but rather must be provided by loading specific proprietary TSRs—device drivers really—on any VM that requires that access.

In this vein, other than for its own proprietary shared RAM drive, you cannot use conventional disk caching programs, RAM drives, nonstandard block devices, or virtual drives with VM386. This limitation effectively rules out the use of many data compression utilities, particularly those utilizing virtual disks. One data compression board, tried in conjunction with VM386, allowed files to be called back and decompressed from the compression boards proprietary virtual disk, but VM386 would not let it save them back to disk.

There are also some other peculiarites you should know about when working with virtual machines. These are not just VM386 issues. In virtual 8086 mode, access to INT13h is barred by the CPU. This is not a problem with VM386's software but rather has to do with general fault protection and access privileges allowed to virtual machines.

For the most part, you might never even notice this. Few programs make INT13h calls. The DOS FORMAT program, however, uses INT13h, as does CHKDSK with the /f (fix) switch; therefore, you cannot format disks while running in virtual mode. You can run CHKDSK on a virtual machine, however. It just can't try to fix the disk if it finds lost clusters, chains, etc. The solution in both cases is simply to boot up in real mode (as you have to at the beginning of a session anyway), do your disk formats and fixes, then boot your virtual machines and go on with your work.

VM386 is a fussy program in some respects. It will only run on an 80386 or higher system with at least a 16-bit bus. Although this does allow it to run on 286s upgraded with accelerator cards, it will not run on accelerator card-upgraded 8088s (unless upgraded with a new 386 motherboard).

You absolutely have to use an enhanced keyboard with VM386. The scan codes on enhanced keyboards are different from the older 84-key types and VM386 just plain will not run—will not even load—unless it finds an enhanced keyboard. Fortunately for those of us who like our function keys along the left-hand side, there are a few enhanced boards on the market that have the function keys where they should be. Northgate makes one and there are several others.

Virtual machines do not have virtual crashes

Sooner or later it's going to happen—who knows why? Sooner or later, you're going to have a virtual machine go down. It is when that happens—more than any other time perhaps—that you will truly appreciate the virtues of the virtual machine. The individual virtual machines seem no more and no less prone to hangups than single tasking on a real machine.

When a virtual machine does crash, none of the other machines, including the host machine, are affected. You simply reboot the crashed virtual machine, like you would reboot any crashed machine, but with the power switch. To date, I have yet to be able to bring the system down by crashing VM386 virtual machines. The isolation between the virtual machines and the host machine is so complete that the system seems almost completely bulletproof.

To quickly put this in perspective, let me stress that, based on several years of multitasking experience, in most cases—with DESQview anyway—one crashed application will not bring down the entire house of cards. If that were the case, I would not be so enthusiastic about multitasking. Still, it does happen. When it does, there is always at least a certain amount of risk.

Because the VM386 host machine is not affected, rebooting a crashed virtual machine usually can be accomplished with just the old three-key (Ctrl – Alt – Del) warm boot. The only caution to observe is that you must be sure the machine you intend to reboot is the foreground machine, because that's the one that's going to reboot whether you want it to or not. If all else fails or you are uncomfortable with the three-key method, the VM386 control menus provide a reboot function, as well.

VM386 is really a pretty easy system to work with overall, with a master menu system never more than a keystroke (Alt – SysReq) away. Multitasking is only the beginning.

More than just one pretty face: the multiuser option

Multiuser systems are attracting increasing attention these days. Traditional networking requires expensive cards and new operating software, which generally involves at least a certain learning curve when moving from a stand-alone platform. Expensive hardware and new software is not necessary with VM386. Unless you're really into doing something exotic, DOS is all you or any user ever really sees.

Going multiuser with VM386 requires only one 80386 with some serial ports available to act as a host machine and some old PCs as terminal emulators or just dumb terminals, which cost about one third as much as stand-alones. The low figure represents only low-end text-only nodes; however, even going toward the high end with graphics nodes, a multiuser system can be far more cost effective than full multitasking.

The two schemes are not mutually exclusive. Multiuser clusters can be networked, providing not only a less costly solution to many office needs but also adding a layer of additional security by allowing multiuser nodes access to networked resources (through a networked host) without actually granting the users at those nodes direct network access.

Particularly when using standalones as terminal emulators, the possibilities are almost limitless in that environment nodes cannot only have their own exclusive printers or other resources but also have access to a shared resource pool, as well. They can even switch back and forth—multitasking if you like—between stand-alone operations and terminal mode access to the host and beyond.

The transition from single-user VM386 multitasking is relatively painless, with various packages available supporting various numbers of nodes. The smallest supports two nodes remote from the host machine. The transition involves little more than installing the new software (which takes about five minutes) and setting up the nodes. If you have already been running a single user VM386 installation, most—if not all—of the configuration files will still be valid on the host machine.

Earlier versions supported text-only applications, except on the host machine. It now is possible to run graphics applications and even multitask on nodes, as well. VM386 has quietly matured into a very powerful family. With its modular design, it exemplifies not only what can be done with virtual machines even starting at the entry level of multitasking but also the range of potential of virtual machine technology.

15
CHAPTER

The MDOS multiuser option

Although users most often normally think of DOS and DOS-like work environments in terms of single user systems, a whole genre of multiuser DOS-based or start-from-scratch DOS look-alikes has evolved, a quiet revolution that has only fairly recently come together under a common banner: MDOS. The name derives from Multiuser DOS. The important thing that MDOS systems have in common is that they allow two or more machines (up to some specified limit) to be interconnected by a simple, inexpensive hardware interface, often sharing applications written to run under DOS.

You might think that this sounds like just another name for networking. Indeed, MDOS systems offer many of the advantages of a LAN, but generally at a substantially lesser price, because MDOS systems require neither the added expense or complexity of network cards in each machine or the high cost of network software associated with some of the better known network systems.

Typically, the only interface required is just a dedicated serial port at each machine. Because the underlying operating system—if not MS-DOS—is something pretty close to DOS, the fact that there is anything else involved is generally invisible to users except, perhaps, at the host machine. For the most part it is invisible there too, once the system is up and running. Besides the lower price tag of most multiuser systems, retraining costs can be reduced to nearly nothing. Ongoing administration is generally much easier than with traditional networking, resulting in further savings.

This does not mean that Multiuser DOS can or ever will obsolete traditional networks. However, in many cases, Multiuser DOS can provide a far more cost-effective alternative. In others, it can be teamed with networking in either new or existing systems to provide the best of both worlds.

Actually, MDOS is nothing new. Only the name is fairly new, coined when what at first was only just an informal group of vendors got together and decided to offer a viable alternative to full-scale networking. The DOS part was and is interpreted rather broadly to include virtually anything that is DOS compatible (i.e., capable of running software written to run under DOS on a machine with an 8086-compatible CPU).

While the design of multiuser systems and the factors determining the effectiveness of the multiuser DOS versus traditional networking are beyond the scope of this book, it does seem appropriate here to look briefly at some of the options and how seamless the transition from a world of isolated boxes to one of connectivity can be. There are also factors to consider that could or should influence both hardware and operating system acquisitions for users who can see the need for connectivity upcoming.

The virtual machine again

In another chapter, I discussed a DOS add-on multitasker called VM386 that took advantage of this special capability, creating virtual machines on the host machine with each virtual machine capable of supporting an application as if it was running on a completely separate 8086-type machine. There, I focused on VM386 mainly as a single-user system, but the basic mechanisms are essentially the same for multiuser systems. VM386 also is available in multiuser increments.

Returning to the underlying mechanics for a moment, however, I showed how this virtualization could be so total that each new "machine" can be configured differently—even booting an operating system different from the one the host machine is running under. This is the extreme case; you cannot virtualize much more than that. You don't even have to go that far to see the virtual machine at work. DESQview even virtualizes hardware, but to a lesser extent. The same is true of Digital's DR Multiuser DOS and PC MOS as well. This area is really nothing new in concept, rather more in application and sometimes degree.

Intellegent Graphics Corporation, developer of VM386, then took the process one step farther in its Multiuser VM386. This companion product simply allows some—or even all—of the virtual machine sessions under its auspices to be conducted outside of the box containing the host, running instead on machines that were, at least within the context of that session, simply dumb terminals.

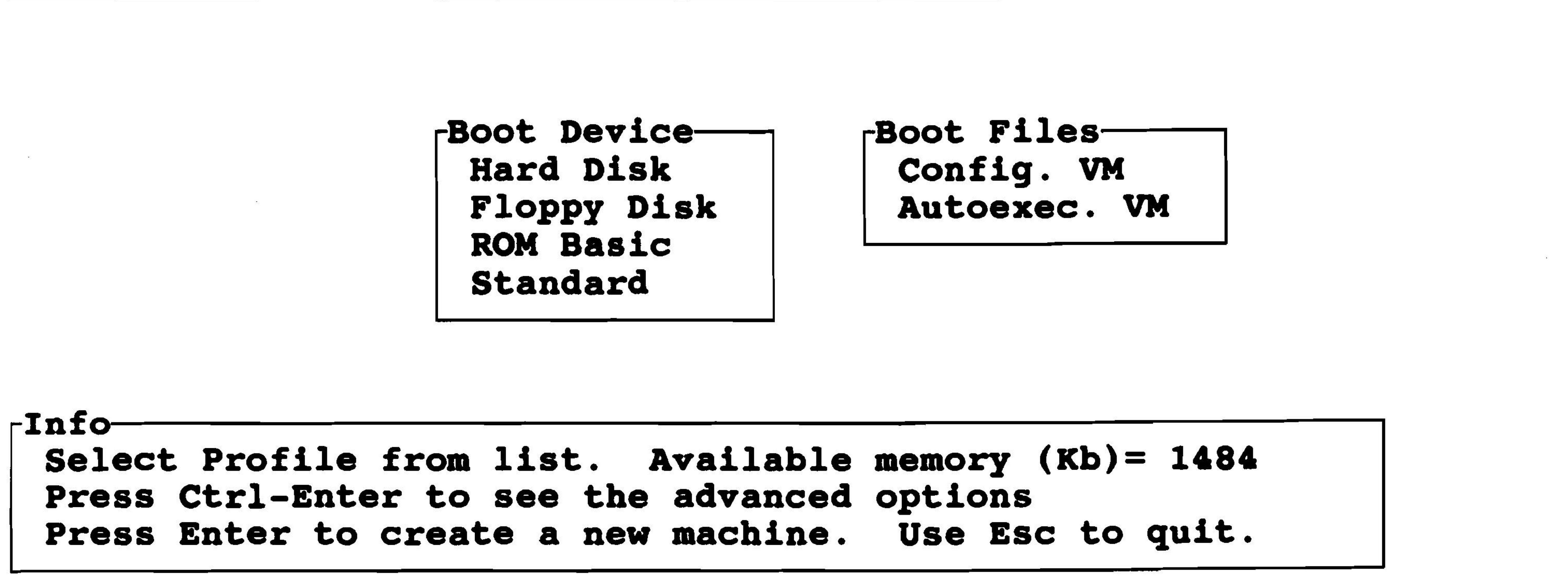
As you can see, it is conceptually a relatively short step to transition from a single-user multitasker to a multiuser system, running, perhaps, the exact same number of concurrent sessions but continuously presenting what would be background sessions on a single user system as active foreground sessions to users at one or more other stations. As a practical matter, there is a bit more to it than that, but at least in principle it is a fairly simple step.

Both the single and multiuser versions of VM386 started with an ordinary DOS system—a bare DOS system—simply loading an additional level, a superset of DOS, on top and running from that. I'll begin by picking up the VM386 story where I left it and taking a look briefly at the ways the multiuser version differs.

Multiuser VM386

This is an ideal starting point for the discussion, because here is a genuine—and logical—upgrade to multiuser status for a product that earned its stripes in the competitive single-user multitasker market. Most of what I said about the single-user version earlier carries over to the multiuser packages, as well. Rather than belabor the basics, I'll refer you back to that discussion.

In particular, I would recommend that you review the discussion of the total virtualization of the hardware by VM386, because in this regard, VM386 (Fig. 15-1) is quite unique from the other multiuser systems that I'll be discussing. Most, if not all, 386 multitaskers—single or multiuser—virtualize the hardware to some extent. That is, they create virtual machines, each behaving like a separate—or nearly separate—8086 machine.



```
1.1]==================== Create Virtual Machine ====================

Profiles          Basic Options                   VM List
 128K_PC           VM Name= 128K_PC                VM1  XYWrite 3+
 256K_PC           Profile= 128K_PC                VM2  DOS
 384K_PC           Base Memory (Kb)= 128
 512K_PC           Foreground Only=  No
 640K_PC           SRM Check= Yes
 COMM              Update Profile= No


                  Boot Device         Boot Files
                   Hard Disk           Config. VM
                   Floppy Disk         Autoexec. VM
                   ROM Basic
                   Standard

Info
 Select Profile from list.  Available memory (Kb)= 1484
 Press Ctrl-Enter to see the advanced options
 Press Enter to create a new machine.  Use Esc to quit.
```

Hardly a pretty face, VM386's top-level control screen is strictly business. Some of the terminology on this and successive screens will send you to the documentation, but the power is there.

VM386's creators have taken this concept to its ultimate conclusion seemingly, developing a DPMI-compatible system that creates and supports virtual machines that are so totally independent of each other that they can run side by side concurrently, even under different operating systems. Building on that, it is, at least conceptually, a small step to physically remove those virtual machines from the host, so as to conduct those sessions from remote keyboards and view them on remote displays.

Moving up from single-user VM386 to multiuser involves a little more than simply loading in the new multiuser software. While the AUTOEXEC.*nnn* and CONFIG.*nnn* files customize individual virtual machines for specific applications, there are other configuration files associated with the host machine. These other configuration files are not transferable, even as they apply to the primary workstation. Instead, they must be re-created for the new multiuser installation, including whatever data is required to configure the system to host remote user virtual machines, as well.

This is a relatively simple procedure, however, and mainly a matter of working through a series of menus. As with the single user version, you also have the option of creating specific virtual machines for the current session only or making them part of the permanent configuration.

Although VM386 does support both extended and EMS expanded memory, it does not currently support the High Memory Area or mapping EMS memory to high DOS addresses, even if the overhead of VM386 system itself is concentrated in the host machine and not shared by the virtual machines that actually run your applications (each of which simply loads its own copy of DOS the same way any real machine would do).

This is not a limitation imposed by the virtual 8086 mode, but rather these are features that simply have not been implemented at this time. However, as a practical matter, the fact that each application runs on a machine specifically tailored to its needs rather than having to load device drivers and TSRs in anticipation of other needs—and the fact that machines running text-only applications can include the memory area above A000h normally reserved for graphics—significantly reduces the need for such support.

On the other side of the coin, whether you are working with real or virtual machines, not all applications need the full 640K that DOS allows (704K or more for text-only applications). To conserve memory, VM386 allows virtual machines to be created in almost any size (the normal menu-option increments are 128K but advanced users can work with smaller increments, if desired). In any case, you have to allow space for DOS (or a reasonable alternative operating system).

The documentation is extensive—some 426 pages for the five-user starter package. All of it is devoted to detailing operations under multiuser VM386 and contains virtually nothing with regard to its workings—much less than is in this book.

Multiuser VM386, however, represents only one approach to providing a multiuser multitasking environment. Its creators chose not to create a total operating system, but rather a perfect superset to DOS (or reasonable DOS look-alike). This need not be a separate step, however, as exemplified especially by DR Multiuser DOS from Digital Research. Introduced in early 1991, DR Multiuser DOS will run on anything from a 386SX up and represents a quantum leap beyond Digital's earlier Concurrent DOS 386, which was discussed in the first edition of this book as one of several interesting possible alternatives to MS-DOS at that time.

It fits the discussion as this point because it does indeed incorporate the capabilities required to support multiple concurrent sessions on virtual machines that are accessible to multiple users as an integral part of an operating system that is almost a perfect MS-DOS look-alike. It also takes a more traditional—and less visible—approach to implementing virtual machine technology.

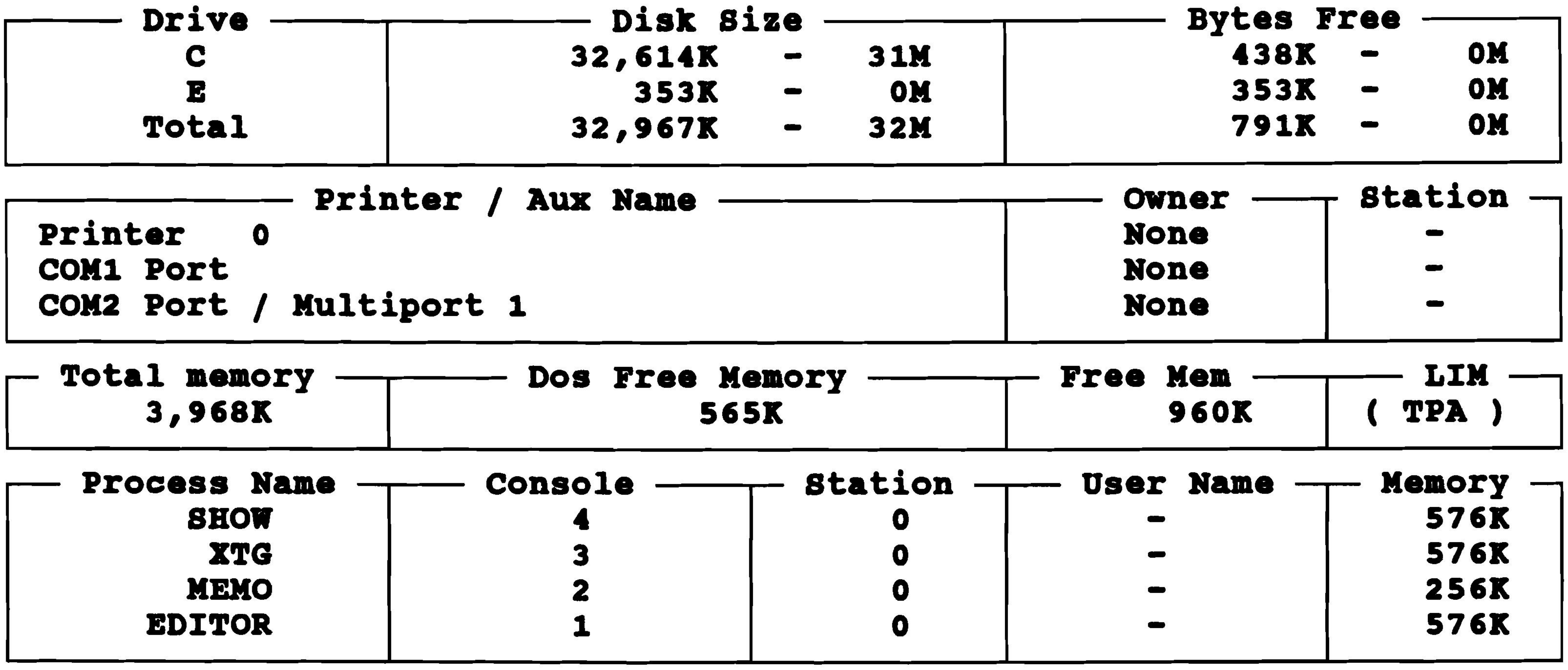
DR Multiuser DOS

Not content with just going head-to-head with Microsoft for the single-user DOS market—a market Microsoft essentially pulled out from under Digital in early PC days—Digital Research now is striving for supremacy in the rapidly expanding multiuser market. Hot on the heels of its single-user DR DOS 5.0 (examined elsewhere in this book), Digital introduced a multiuser product, DR Multiuser DOS 5.0, which started shipping in the second quarter of 1991.

DR Multiuser DOS is a big operating system. It has to be, given the fact that it theoretically has the power to support up to 256 users, all running multiple sessions. As a practical matter, most DR Multiuser DOS systems probably will not have more than 10 users, with each running up to a maximum of eight applications concurrently, but the power is there.

The key program, a hidden file called DRMDOS.SYS, is 245,504 bytes long or almost four times the size of MS-DOS 5.0's two hidden files put together. By breaking the kernel into pieces and fitting most of them above 640K, however, the net conventional memory space left to run applications can run as high as 592K, although it typically will run a little less as shown in Fig. 15-2, which represents a fully configured system with VDISK and disk caching that still has up to 576K for each application on the main console. Intriguingly, a PC with a CGA card used as a terminal can have even more for each of its sessions than the host machine with a VGA card.

Although descended from its earlier Concurrent DOS and reminiscent of it in many ways, this is essentially an all-new product geared exclusively to today's world of 32-bit machines. This means that, although individual workstations can be anything from dumb terminals or old 8088 PCs on up, the host machine can only be a 386 or higher. It is essentially a 32-bit operating system that avails itself of 32-bit processing power while presenting a conventional DOS environment for



```
┌──── Drive ────┬─────── Disk Size ───────┬──────── Bytes Free ────────┐
│       C       │   32,614K  -   31M      │      438K  -      0M       │
│       E       │      353K  -    0M      │      353K  -      0M       │
│     Total     │   32,967K  -   32M      │      791K  -      0M       │
└───────────────┴─────────────────────────┴────────────────────────────┘
┌────────────── Printer / Aux Name ───────┬──── Owner ───┬─ Station ─┐
│ Printer   0                             │    None      │     -     │
│ COM1 Port                               │    None      │     -     │
│ COM2 Port / Multiport 1                 │    None      │     -     │
└─────────────────────────────────────────┴──────────────┴───────────┘
┌─ Total memory ─┬─── Dos Free Memory ────┬─ Free Mem ───┬──── LIM ──┐
│    3,968K      │         565K           │    960K      │ ( TPA )   │
└────────────────┴────────────────────────┴──────────────┴───────────┘
┌─ Process Name ─┬── Console ──┬─ Station ─┬─ User Name ─┬─ Memory ──┐
│        SHOW    │      4      │     0     │      -      │     576K  │
│         XTG    │      3      │     0     │      -      │     576K  │
│        MEMO    │      2      │     0     │      -      │     256K  │
│      EDITOR    │      1      │     0     │      -      │     576K  │
└────────────────┴─────────────┴───────────┴─────────────┴───────────┘
```

15-2 Memory report of a fully configured system running under DR Multiuser DOS shows up to 576K available to each session running on a VGA-equipped host machine.

applications software, both at the host machine and everywhere along the line.

To someone used to multitasking in a single-user DOS environment, the difference should rather quickly be apparent (although in fairness, there are other factors that enter into the equation as well). Still, it is not how cleverly the system time slices or whether it is fully utilizing the 32-bit potential of the CPU that matters. What matters is the bottom line: performance. This one is impressive.

The double whammy

DR Multiuser DOS can be installed as the sole operating system on the host machine, and no doubt would be in most cases. However, it also can be co-installed in a dual-boot configuration on a hard disk that already contains a bootable DR DOS, MS-DOS, OS/2, or any of several other operating systems. Unlike PC MOS, DR Multiuser DOS dual-boot installation does not require repartitioning your hard disk. DR Multiuser DOS can coexist quite nicely in the same primary disk partition in much the same way that various OS/2 and MS-DOS releases have been able to coexist.

However, in this regard, it is a whole lot easier than setting up an MS-DOS/OS/2 system. Only certain MS-DOS releases can be teamed with certain OS/2s. With DR Multiuser DOS, it doesn't matter (which is a good thing, because unlike most other operating systems, DR Multiuser DOS can be installed only to a hard disk). You can create a bootable DR Multiuser DOS floppy directly from the distribution disks; however, you can only configure a hard disk installation via the menu-driven Install/Setup utility.

Once it is installed in a dual-boot situation, the following message appears on screen whenever you boot the host machine:

Load Multiuser DOS (Y or N)?

An affirmative response loads the DR Multiuser DOS operating system, while a negative response executes a special loader for whatever other system you might have had in place when you installed DR Multiuser DOS.

However, you will have only a single and somewhat larger AUTOEXEC.BAT file than you started with that then serves both, invoking functions unique to DR Multiuser DOS only when you're running under DR Multiuser DOS and ignoring them when you're not, as you can see for the excerpt reproduced in Fig. 15-3. In a dual-boot situation, you probably will have to do some juggling and probably a little tailoring.

```
:OSLOAD
@ECHO OFF
if "%os%"=="DRMDOS" goto OSBEGIN
.
.
.          (DOS portion of AUTOEXEC.BAT)
.
.
@echo off
:OSBEGIN
@ECHO OFF
REM The OSBEG and OSEND labels tell the SETUP program which
REM statements it should process. Put any additional
REM statements for REM Multiuser DOS between these two labels.
REM Any other statements eg. for other operating systems
REM should be placed outside the labels.
PATH C:\OSUTILS;c:\dos\batch;c:\dos\utility;c:\bisex\utility
APPEND C:\OSUTILS
NETDRIVE C: /R
SUSPEND = OFF
IF "%CONSOLE%"=="" SET CONSOLE=%3
PROMPT %CONSOLE% $p$g
:OSEND
```

15-3 Excerpt for an AUTOEXEC.BAT that serves both DR Multiuser DOS and (in this case) MS-DOS 5.0 in dual-boot installation. Although lines unique to DR Multiuser DOS are created automatically during installation, integrating them with the original file might require some fine-tuning on your part.

DR Multiuser DOS does not change the existing CONFIG.SYS file on a dual-boot installation. However, the DR Multiuser DOS installation program does create a special file, called CCONFIG.SYS, exclusively for DR Multiuser DOS. Whichever operating system you load then runs its own. Here, as in a number of areas, Digital has conspicuously not carried over features from its single-user DR DOS in this multiuser release, although this is a minor point.

As an aside, the Install/Setup utility mentioned above is interesting. A large program—almost 120K—SETUP.EXE, which is needed any time you want to change your DR Multiuser DOS configuration, does not exist on the distribution disks. At some point during the installation, it is created from INSTALL.EXE, which remains on the distribution disk in its original form while SETUP.EXE is copied to your hard disk. From that point, DR Multiuser DOS recognizes INSTALL in the same sense as it is used in either MS- or DR DOS: to load TSRs from the CCONFIG.SYS rather than from the command line or batch file later.

Aside from configuration or reconfiguring the system in general via the Install/Setup utility, individual DOS sessions for individual workstations are set up with simple START*xxy* batch files. However, each session on each station must have its own individual startup file.

On the other hand, it is one of the easiest systems I've seen. On startup, DR Multiuser DOS reads and automatically executes whatever START*xxy*.BAT files it finds, in much the same way as DOS looks for and executes the AUTOEXEC.BAT file. In your root directory, you might have something like this:

```
START001.BAT
START002.BAT
START003.BAT
START004.BAT
```

Four is the default number of sessions on the host or at each workstation, but this can be increased to up to eight for each, as required.

A number of usable sample batch files are provided on one of the distribution disks (but are not automatically copied to your hard disk). The documentation also deals extensively with creating superuser (the host machine on multiuser system) files for various typical applications. However, in many cases, simple batch files are all that is required, often containing nothing more than the command required to open the program, such as C:\SUPRCALC.SC5. They also can be full-fledged batch files that change directories, establish a different path, or append for that applications, etc.

Using proprietary commands, these batch files also can be used to establish other than default values for the amount of memory allocated to that session, the amount of EMS memory available to it, and other values specific only to that window, or session, as windows are referred to. In any case, for anyone familiar with batch files, it is a breeze.

DR Multiuser DOS provides a choice of three different hotkey methods for switching from one session to another, any of which can be enabled or disabled at the user's option when configuring the system:

Ctrl–*n*	to switch directly to a session by its number
Alt–Esc	to browse through open sessions
Ctrl–Esc	to call a pop-up window showing the status of all sessions

These are simply defaults, however, and can be changed during the installation (or at any later time) if these conflict with other keyboard assignments.

The rather risky option of being able to reboot the host machine—*ergo* abruptly terminating everybody—via the old three-key (Ctrl–Alt–Del) is enabled by default, but can be disabled at the user's option when configuring the system.

One of the factors that contributes significantly to the ability of DR Multiuser DOS to mask the fact that it is actually not only managing multiple concurrent sessions on the host machine but possibly on several other PC/terminals is the way that it divides the CPU's attention between them. This becomes a much more important consideration in multiuser systems than with single-user multitaskers by sheer weight of the number of sessions the host machine might have to duplicate—in this case, up to eight times the number that would be involved if every user had a single-user multitasker.

Idle power

One of the keys to DR Multiuser DOS's success in this area is an IDLE feature that can be set either to Off or On. The default is On, which means that, during periods when there is no keyboard input, running applications are skipped over, so their time slice can be put to better use. All decent multiuser (even single-user) multitaskers do this to varying degrees. However, DR Multiuser DOS seems to be among the best I've seen so far, although—as with any of them—improper configuration can quickly rob you of performance. IDLE OFF is one of the options that can be set for a particular session by including it in the START*xxy*.BAT (or entered from the command line before opening an application manually), when needed. Realistically, about the only time you should ever need to change the default is when you want to keep on crunching numbers in the background—or sorting a database, etc. The option is readily available when needed.

Security is often a serious concern on any system accessible to other users or any one with a little smarts and an unhealthy curiosity who happens to come along when no one is looking. DR Multiuser DOS provides a comprehensive password protection system with three separate levels:

- Read (r)
- Write (w)
- Delete (d)

These can then be assigned to control access by:

- The owner
- The group
- The world at large

Access rights can be set up with the XATTRIB command. A group of files then might have their levels of protection setup something like this (dashes indicate that access is not allowed):

```
rwd ---  ---   fred     doc c:\usr\fred\news.doc
rwd rwd  ---   joanne   doc c:\usr\fred\joanne.txt
rwd ---  ---   fred     doc c:\usr\fred\personal.doc
rwd rw-  r--   fred     doc c:\usr\fred\letter.doc
```

Other security utilities include one you can use to lock access to your machine during coffee breaks, etc. Overall, it seems to be a well thought out scheme.

In terms of overall memory management, DR Multiuser DOS provides full EMS support for DOS applications. DR Multiuser DOS can run Windows sessions in real mode. It does not support either the VCPI or DPMI specifications, however, so it cannot run any DOS-extended applications. With its dual-boot capability, it does not completely foreclose that option either. Overall, DR Multiuser DOS is a powerful contender for the rapidly expanding MDOS market. However, it is not alone.

PC-MOS

Another player in the game is The Software Link, Inc. It has been in the multiuser DOS look-alike business for sometime, with a succession of PC-MOS products. Intended at one time strictly for the 80386 market (PC-MSO/386), the product now embraces the use of 80286 machines as servers with its fourth generation multiuser/multitasking operating system, now simply called PC-MOS. In it, special support is provided for 286 machines equipped with the All ChargeCard (see chapter 16).

MOS stands for Modular Operating System to distinquish it from DOS. However, it is , at least outwardly, a pretty good look-alike, with a command structure that is pretty similar to DOS. The "Modular" in the name is meant to be more than just an excercise in semantics. The basic package is designed for single-user multitasking, with expansion modules available to expand the basic installation to accommodate groups of workstations. The interest here is mainly in the multiuser/multitasking operating environment.

A number of functions that are external to MS-DOS or DR DOS are internal in PC-MOS. PC-MOS is structured quite differently from DOS, with most of its bulk (122,900 bytes in version 4.1) contained in $$MOS.SYS and another 33,207 bytes in $$SHELL.SYS. COMMAND.COM, on the other hand, is only 20 bytes long.

While the size of the PC-MOS kernel might seem overwhelming at first glance, it is really hardly any larger than—if it is as large as—the total overhead of DESQview loaded on top of DOS. The exact balance depends on which versions of DOS and DESQview you want to talk about. The PC-MOS kernel is written to allow a good part of it to load above 1024K, in what would be the HMA if MOS used an XMS-compliant driver, and allows still another piece to be loaded in upper memory on 386*s*, *i*486s, and ChargeCard-equipped 286s that support it. As is the case with DESQview on comparable machines, the net result is something anyone can live with.

Overall, the command structure of PC-MOS is similar to DOS in many ways, with many commands having identical names and syntax. At first glance, the syntax looks a little different than MS-DOS, with the documentation showing a period preceding all of PC-MOS's otherwise look-alike commands. The prefix dot is actually an option that can be used, if needed, to prevent conflict with other software that might respond to the same commands if they are not prefixed. This is not considered to be a common problem, however. PC-MOS itself defaults to DOT OFF unless specifically set to DOT ON.

Despite the fact that many of the commands are the same, many others are not, so there would seem to be more of a learning curve involved than with some of the other multiuser systems, particularly as far as the system administrator is concerned. This seems especially true during the initial setup stage, where, although things are generally well-documented in the manual, the information is not always at your fingertips.

Although marketed primarily for use with 80386s, PC-MOS/386 also can be used to advantage with 80286s, particularly those equipped with the All ChargeCard or other memory management card that adds remapping to the otherwise restricted 80286 chip. Even without enhanced memory management on an 80286, PC-MOS can relocate a part of its kernel in extended memory above 1 Mb with the help of a special device driver.

PC-MOS also can be run on 8086 and 8088 systems. At a glance, that might not seem important; however, because 8088s can be used in PC-MOS port-to-port multiuser configurations, it is helpful (although not mandatory) to have all users on the system at least using the same operating system.

One of the features that is more or less unique to PC-MOS is that it allows users to call up each other's user sessions, not just the same applications and files, on their displays. Access is controlled, so this would seem to have a number of practical applications.

Here, the rules seem to get a little nebulous, however, As mentioned above, only the workstation that booted the application in the first place has access rights that allow rebooting the virtual machine that the application is running on. However, if that workstation is powered down—actually switched off—the virtual machine (which actually only really exists within the host machine) remains alive and the work session still is accessible either by other users or by the original owner, if that terminal is brought back on line.

Configuring PC-MOS is semi-automated by a menu-drive ACU feature. In view of the number of proprietary device drivers and options, however, configuring PC-MOS will keep you busy for awhile. Also, unless you want to go with the defaults, you might not have the answers you need right at hand. While the automated routine puts a rather ominous looking header on the CONFIG.SYS that it creates, as seen in Fig. 15-4, it is purely informational and has no bearing on the system.

Fine-tuning PC-MOS takes time. Some of your DOS CONFIG.SYS commands especially might need to be eliminated (FASTOPEN for instance) and some others unique to PC-MOS might need to be added.

To utilize the capabilities of 386s and *i*486s and ChargeCard-equipped 286s, PC-MOS supports remapping unused address between 640K and 1024K with EMS memory. This is accomplished through a FREEMEM command that, when added to the CONFIG.SYS, can point to as many as five noncontiguous blocks that PC-MOS can use up there. Otherwise, memory management is automatic under PC-MOS.

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+ This configuration file created by TSL Auto Configuration Utility.
;+ Do NOT remove or modify this file heading.
;+ Built:  5/29/91 @ 20:54:49
;+
;+ CONFIG-INFO: MAX-TASKS=6
;+ CONFIG-INFO: SYSPATH=A:\
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

15-4 The CONFIG.SYS header written by the PC-MOS automatic installation program is purely informational and, a with any lines beginning with semicolons, is ignored when CONFIG.SYS is read by the system.

Now you see it, now you don't

One of the unique features of PC-MOS is that device drivers can be loaded either from the CONFIG.SYS (in which case, they become global and part of the environment inherited by every virtual machine) or they can be added later. In the latter case, they are local, affecting only the environment of the virtual machine for which they were added. PC-MOS also provides a means of removing device drivers that have been installed, but that might not be required for the remainder of that session.

This set of features essentially stands in sort of a middle ground position, between something like VM386 at the one extreme, which does not recognize any device drivers globally (except for a few proprietary ones, such as those required for access to essential shared system resources such as ports, etc.), and a DESQview-type situation, in which all applications must share a common environment.

Figuring out which ones should be global and which ones shouldn't be comes under the category of fine-tuning. You have to be a little more careful about using device drivers with PC-MOS than with some other systems; however, because the work area—the amount of conventional memory available to run our applications—is somewhat smaller with MOS than other multiuser systems.

The first thing you probably should do is to check to see how all of your applications run under PC-MOS while you're still under PC-MOS's unique 30-day money-back guarantee. For that, you can get PC-MOS up and running pretty quickly on the host machine, booting directly from the distribution system floppy, if you don't want to risk your current DOS hard disk configuration, while you experiment with PC-MOS.

Compatibility with some applications software was a problem with earlier PC-MOS releases, but a check of some of those known to have had problems in the past would seem to indicate that, by release 4.1, the specific problems that I was looking for had been overcome. In any case, you have ample opportunity to find out, because PC-MOS comes with one of the best and most foolproof money-back guarantees in the business.

Sold with a money-back guarantee

You can reboot as often as you like and try any or all of PC-MOS's features on your own machines and at your leisure. You not only can test your applications software, but you also can fully configure your system and try running a complete multiuser setup under PC-MOS without voiding the 30-day guarantee.

The gimmick is that, as it comes out of the box PC-MOS, will run only about an hour, then shut down. You can reboot as often as you like, but each session can last only an hour, until a little built-in timer shuts you off. However, if at any point, you're satisfied and ready to adopt PC-MOS, there is another disk. This one is in a sealed envelope. Opening that envelope to get the disk that turns the timer off and gives you a permanent PC-MOS terminates the money-back guarantee.

This try-it-before-you-buy-it policy holds true not only for the core of the operating system itself, but also for additional software modules, like Software Link's LANLink networking package. It's too bad some other software companies don't pick up on this idea.

There is a little more than altruism in the approach The Software Link has taken here, however. Although the process is reversible, the only way you can create a bootable PC-MOS disk is by rewriting the boot sector of the disk. A disk formatted by DOS cannot be made bootable under PC-MOS except by rewriting the boot sector, which then makes it nonbootable under DOS. While there might be some advantage to taking this approach, it does create at least a nominally nonstandard situation.

To offset this, PC-MOS does provide a special mechanism for configuring a disk in a way that will allow it to boot under either DOS or MOS. However, to do this require repartitioning your hard disk and dedicating at least one partition to PC-MOS, because it requires a special proprietary boot sector to be bootable. Repartitioning, takes you right back to square one with your hard disk, wiping out everything on it, so this is not something you want to jump into hastily.

Unfortunately, the up front installation documentation only talks about replacing DOS with PC-MOS on your hard disk (which can be done without repartitioning and disturbing any of your other files or software) or about repartitioning a hard disk to set up the dual-boot capability. However, you can use the MSYS *d*: command to write a new boot sector to a floppy or use the PC-MOS formatter (both options are documented elsewhere in the manual) to prepare a floppy that you can play with and configure. I made up a 1.2 Mb floppy that way that was not only bootable under PC-MOS but had all of the utilities contained on the three distribution disks with space to spare. With something like that, you can fully configure your system without disturbing your hard disk DOS, until and unless you're fully satisfied that PC-MOS is for you.

Overall, PC-MOS is a rather large program, not in terms of the amount of memory it takes up—which is rather modest—but in terms of the number of drivers and utilities provided. It's a rather complicated package to set up—harder than Multiuser VM386 or DR Multiuser DOS. Still, it is a powerful tool that offers a variety of unique capabilities that deserve careful consideration, because it is the long-term capabilities that matter most.

Only the tip of the iceberg

I only have scratched the surface of the world of the multiuser system. In confining discussion to DOS-like operating environments, I have completely overlooked such powerful multiuser systems as XENIX, which is sort of the granddaddy of multiuser systems, and this is not entirely fair. While DOS and UNIX represent quite different operating platforms, with a little software help, the two can coexist quite nicely.

With products like SCO VP/ix, the files can all be mixed together. SCO VP/ix manages any piping or redirection between DOS and XENIX files or processes that might be required. You can start a session under DOS or under XENIX; it doesn't really matter. With operating environments like X-Windows and the new DESQview X, you don't even have to have a 386 or higher to make it one big almost-happy family. If the DPMI specification lives up to some expectations, compatibility issues will slip even farther into the background.

The memory issues that are the primary focus of this book are handled differently. What XENIX looks at as standard memory is essentially what is called Extended memory in the context of the DOS environment: Linear memory starting at 0000h and going right on up. The minimum recommended for a XENIX configuration is 2 Mb. Really, a minimum of 4 Mb is recommended with still another megabyte for each additional SCO VP/ix user on the system.

What you call memory is of little importance. What matters most is what you can do with it. Under XENIX or most any operating system today, things like EMS or whatever kind of memory you need can be emulated. Even the operating system is important only as a conveyance, an interface. What really matters is your applications software—the stuff that does the work that makes it all worthwhile, no matter how many of you are all hooked up together.

What you don't see

There are some things you don't see here among the multiuser options. All of the systems I have discussed run exclusively on 396 or higher systems—or if not exclusively, by preference. Yet none of them provide or allow for the kind of EMS memory management that is taken for granted with single-user 80386 systems. It appears that, while all (or at least most) of these systems use upper memory for their own use (Multiuser VM386 seems to be an exception), they do not make any upper memory available for other uses.

You cannot help them out with QEMM, ALL Charge 386, 386MAX, or any of the powerful memory managers I have discussed in this book—or even any that I haven't—because all of these multiuser packages are total packages in that they have their own built-in memory management tailored to their own unique and special needs. They can work only when given total control over all system memory resources. They do it their way, or not at all.

This means that any device drivers—other than any that may be internal to these various systems—must load down in conventional memory. You might have to reexamine your present use of upper memory and do a little weeding out wherever possible, although both PC MOS and VM386 have workarounds that help ease the problem.

Also, there are some device drivers that, for various reasons, some or all of these multiuser systems cannot work with. VM386, for example, (single or multiuser) cannot handle nonstandard block devices, such as the virtual disks created by most real time data compression utilities. I have found some others with some of the other systems too, not to pick on VM386 unfairly.

However, these are things that users can live with. Face it, it's been only in the past several years that anyone has had upper memory, reserved memory, or high DOS memory—whatever you want to call it—to work with anyway. Call it a small step backward if you will; however, that is a small price to pay for the giant forward leap that multiuser multitasking (MDOS) represents.

16
CHAPTER

Keeping up, or trying to

At some point, sooner than they'd like to think, most users have to face the question of upgrading their overall systems. It is a normal, healthy stage that, in general, marks at least a fair degree of maturation, both in terms of needs, skills, and attitudes about computers. Once that is determined, the issue then breaks down not to the need to upgrade to keep pace, but rather to which one of three possible avenues to pursue when just adding memory is not enough:

- Perform a partial system transplant via accelerator cards, while retaining the same motherboards and hardware otherwise.
- Replace the motherboard, but otherwise retain all or part of your old hardware.
- Start from scratch, replacing your current CPU with complete brand new 80286, 80386, or 486 systems.

The options in themselves have not changed greatly since the first edition of this book was published (some of the product names have fallen by the wayside as might be expected, but there are others in their place). However, in the interim, the industry has changed, not just in measured steps but quantum leaps. At this point in time, the picture that emerges is quite different from what it was not too long back. The first of these options represents both the cheapest and the most expensive solutions, also the easiest but not always the best. Option two is more middle-of-the-road, often actually requiring more careful thought than the purchase of new equipment or merely adding on more memory. However, either of these first two generally represent the least expensive ways around the problem.

None of the above solutions are automatically the best. Just throwing money at the problem and buying all new hardware might produce the poorest return—or the best, depending on the specific needs and just how well whatever new equipment you purchase fits that need, not only for this year but for the next few years as well.

For my purposes here I've put the purchase of complete new systems last for a couple of reasons. If you have existing hardware, prudence—and the bean counters in most corporate situations—dictate not just rushing out and buying something else before you check out all the options. By the time I've gone down through the pros and cons of the other options offered, I will have touched on many of the issues that should go into buying new hardware.

The questions are—or at least should be—the same in any event. While, with the exception of some products that have come and gone, the options themselves have not changed significantly over the past couple of years, the world has changed drastically. With those changes, some rethinking might well be in order now before reaching a decision. So, I'll start then at the easy end of the scale and work my way from there.

Supercharging that old 8088

Generally thought of as the cheapest, least traumatic upgrade options, there are several accelerator cards on the market. Accelerator cards physically replace the original CPU chip with something higher, bringing with them their own clock and often their own memory, plus whatever other support hardware might be necessary to complete the conversion.

The popularity of this kind of upgrade has declined considerably, however, due in part to the fact that, by now, most serious candidates for upgrade already have been upgraded, phased out, relegated to serving as dumb terminals on multiuser systems served by 80386 or 486 hosts, or just plain died. In any event, with this decline, there has been a corresponding decrease in the number of accelerator cards to choose from.

The typical accelerator card replaces the original computer chip with a higher-powered, faster chip—an 80386 in the place of an 8088 or in the place of an 80286 to upgrade AT-type machines. However, because the problems and available options for upgrading 286s are quite different than starting with an 8088 machine, I'll look at the 286 situation just a little later on.

For upgrading 8088 to 386s, one of the most successful—and durable—accelerator cards is Intel's 16MHz InBoard 386PC. Intel couldn't have thought of a better way to increase the market for their chips without stepping on toes. While for several years they also made a card that converted 286s to 386s, their 8088-to-386 card produced a more spectacular improvement at lower cost.

Priced at just under $800, the PC version can make an old 8088 run about four to seven times faster on average, or about twice as fast as an AT. It also has a socket for an optional 80387 coprocessor, if you need one. The whole package—even including one of Intel's optional piggyback memory expansion modules—only takes up one of your precious expansion slots. It is recommended that you have at least a 125 watt power supply in your PC to support it, but many clones come with adequate supplies. If you have a genuine Big Blue, you've likely long since replaced the anemic 65 watt supply that IBM supplied, anyway.

Accelerator cards have a definite advantage when it comes to ease of installation. Typically, you simply remove the old CPU chip from your machine—a bit of brain surgery, but not at all difficult—slip the accelerator card into an empty expansion slot in your machine, and plug an adapter cable into the old CPU socket. You're off and running. Even including adding the required device driver to your CONFIG.SYS, installation can be accomplished in well under an hour—probably as little as 20 minutes for anyone with some experience installing boards in any of the expansion slots.

Suddenly that old 8088 PC isn't just a PC any more. It's a full-blown 386 sizzler—or is it? The answer is: not entirely—though for many applications it comes close enough. What it doesn't have is a 16-bit bus comparable to a factory-new 286 or higher machine (except when upgrading a 286 to 386 status). To get around that, Intel also offers piggyback memory boards that provide up to 4 Mb of full 32-bit access extended memory—memory that also can emulate full LIM 4.0 expanded memory—bypassing the bus issue except for I/O services.

In addition to the limitations of the 8-bit bus, the access rates of hard disks installed in most 8088 machines are significantly longer than for the generally more expensive hard disks sold for faster machines. For example, in a side-by-side test using the new task swapper in DOS 5.0, it took 25 seconds to swap an application to the hard disk of an upgraded PC, only four seconds to do the same exact job on a new factory-built 386 SX running at the same clock speed. (The hard disks in both cases had been unfragmented immediately prior to the tests and the interleave factors were checked for optimum performance.)

I/O, disk I/O especially, can be a real bottleneck, especially when starting with an 8088. However, considering the number of applications being tailored to run in 32-bit extended memory using DOS Extenders today, that might be less of a drawback than it once was. Those programs ideally try to leave DOS and all such problems behind as fast as possible and keep I/O to a minimum. The better they can manage that, the less these other factors should seem to matter.

One little quirk you're sure to find annoying is the length of time it takes to boot your altered system—especially if it is, or was—a PC. Accelerator cards do not exist—as far as the motherboard is concerned—until the system is nearly finished booting and until the CONFIG. SYS file loads the device driver that makes it recognizable. The original ROM BIOS boots the system, which means it has to go through the whole slow memory checking exercise, etc. Then the InBoard has to go through a little dance of its own. It takes just about two minutes before it wakes up enough to even look for the AUTOEXEC.

Still, I used an InBoard-upgraded 8088 PC on an almost daily basis in my office for several years—much of the first edition of this book was written on it—regularly running even CPU-intensive programs like AutoCAD. It certainly did its job.

When considering going this route, however, remember that you are starting with an old machine that, in a worst case scenario, might already have as much as six or seven years of hard use on the old motherboard and whatever supporting chips the installation might require. Given this, unless you're starting with a fairly new machine, I really wouldn't recommend this kind of upgrade anymore.

The 286 dilemma

Not surprisingly, you can upgrade 286 AT-type machines to full 386s, too. Such upgrades generally will outperform any upgraded PC/XT class machine (short of replacing the motherboard) because:

- The base machine has a 16-bit bus
- Most AT-class machines are fitted with faster access hard disks

This option has had its ups and downs, however. It was popular when the cost of brand new 80386 machines was high, then fell into general disfavor with the introduction of the 80386SX and generally declining prices for new machines. For a time, the cost of upgrading an 80286 was often significantly higher than the cost of upgrading an 8088 to 80386 status. (The Intel InBoard for 8088s retailed at $995, its InBoard for 286s sold for $1295.)

As plunging prices made complete replacement 386SX motherboards readily available at mail order prices (starting at $400 or less), this ceased to really be a viable option. One or two vendors have continued to sell accelerator cards for 286s at significantly reduced prices. Intel, however, withdrew its InBoard AT from the market.

The ALL 386SX: a new approach

Into this void came ALL Computers, Inc., a Canadian firm. Already widely known for its ALL ChargeCard, another update option I will discuss later in this chapter, ALL developed a new series of plug-compatible adapters. Unlike the ChargeCard, which simply added additional hardware support to the original 80286 chip but did not replace it, this new line, starting with the ALL 386SX, actually replaces the 16-bit 80286 with a 32-bit chip. Prices start at just under $400 suggested retail, including ALL's potent ALL CHARGE 386 memory management software, which was recently upgraded to include automatic optimization.

These differ from the traditional accelerator card in that, rather than plugging into the bus and taking up an expansion slot, these palm-sized packages plug directly into the CPU socket. Rather than bringing their own clock with them, they use the existing clock, running at whatever speed the original machine was designed to run at, eliminating mismatch problems, adding wait states, or other work arounds.

One of the big advantages that the ALL 386SX (and similar) upgrade packages enjoy over accelerator cards is that the microprocessor chip accesses whatever memory is installed on the motherboard the same as it always did. All that changes is what you can do with it.

This is significant because accelerator cards often require that system board memory be disabled, replacing it totally with their own, and letting the system board memory go to waste. I already had 4 Mb installed on my 286 when I upgraded it a while back. While it's never a problem finding a happy home for left over 1 Mb SIMMs, the less juggling and dislocation the better.

The level of expertise necessary to install a ChargeCard is about on par with that required to install any expansion card—and certainly less than that required to replace a motherboard. Once the cabinet is opened, installation requires little more than removing the 286 chip, which is easily identified, and plugging the small, palm-size ChargeCard circuit board into its socket. The 286 then is plugged into a socket on the ChargeCard.

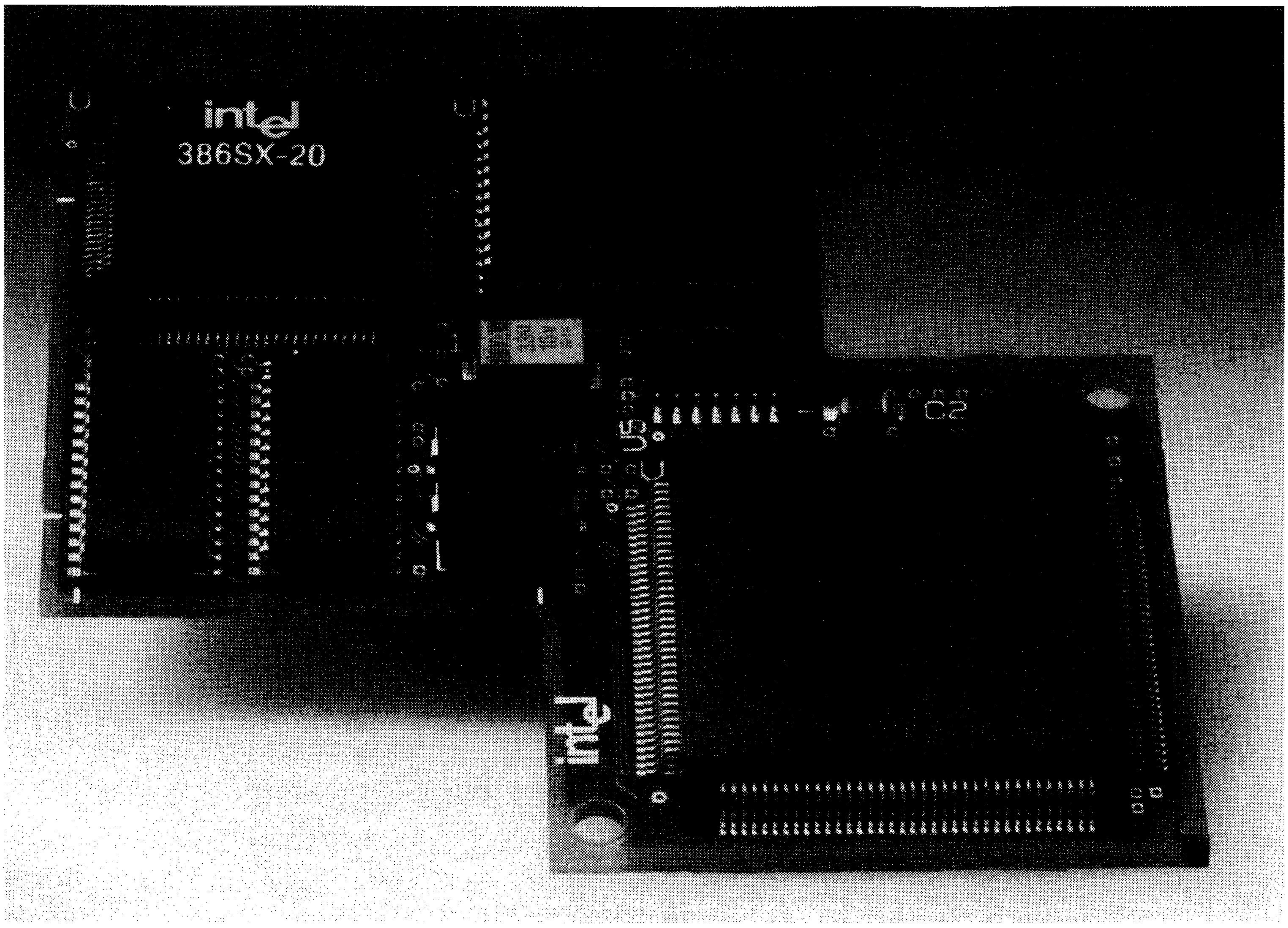
This brings affordable 32-bit processing and the ability to run all of today's new 32-bit DOS-extended software within the reach of almost everyone. ALL already has announced that it is working on still other, hotter plug-in upgrade modules—up to and including one that mounts an *i*486. Up just got a little higher.

The empire strikes back: Intel's SnapIn 386

Now, Intel's back in the game, taking a somewhat similar tactic with its new SnapIn 386 module (Fig. 16-1). It replaces the original CPU with a 386SX chip. It also brings along its own 20 MHz clock and a custom integrated circuit chip containing support logic for the upgrade module's 16K SRAM cache, plus support for the unit's 287 math coprocessor interface, all for $495.

In addition to opening the door to 32-bit processing, making it possible to run anything up to and including today's most powerful 32-bit DOS-extended software, users can expect as much as a two-times increase in performance, depending on the specific applications being run.

That last item, support for a 287 math coprocessor should be of special interest to many 286 owners, because having to add in the cost of upgrading their math



16-1 Intel's SnapIn 386 module converts 286 PS/2s to genuine 386SXs while maintaining support for 80287 coprocessors. This is only one of several relatively inexpensive plug-compatible upgrade modules that have largely replaced accelerator cards for the 286 market.

coprocessor as well might weigh heavily in making the decision. Suddenly, the upgrade option just got more interesting.

However, there is a catch. Initially, these Intel modules are sold only for model 50 and 60 IBM PS/2s and the use of this module with any other machine is not supported by Intel. However, still another company, Kingston Technology, is already marketing another plug-in compatible upgrade module for 286s. Clearly this is a hot area.

Charge it with the ALL ChargeCard

Until now, one of the most interesting pieces of upgrade hardware in the marketplace for users with 80286 systems is not an accelerator card but rather a sophisticated memory manager for the unruly 286. At prices starting at under $200 it still is. This is the upgrade route I went on my in-house 286.

Up until now, I have talked about memory managers strictly as a software function—installable device drivers that often provide other functions beyond just those required to fully implement the LIM 4.0 EMS specification. ALL Computers, Inc. has gone one step farther, providing added hardware support for the underlying 80286 microprocessor chip—and at a significantly lower price than other current upgrade options.

As discussed earlier, software alone can do little to unlock any hidden capabilities of either the 8088 or 80286 chips—mainly because there is little hidden to develop. Just as either will support a math coprocessor that can crunch numbers faster and more efficiently, it is possible—particularly with the 80286—to add devices that support other functions through auxiliary hardware. The ALL ChargeCard 286 adds full memory remapping capabilities otherwise unique to the 80386. You can even run most 386-specific DOS-extended software on charged-up 286s.

Starting with the raw 80286 and designing around it, the ChargeCard 286 actually can map memory into all of the addresses from 640K right up to 1024K—right on through the 64K block of addresses reserved for ROM if you want it to. This is full LIM 4.0 EMS support that can map memory to any legitimate DOS address (providing the underlying hardware supports it).

By default, the ChargeCard and ALL's proprietary software steer clear of any ROM regions, as well as areas reserved for video memory—although it's usually smart enough to determine if your display actually needs the entire area normally set aside. With only a CGA adapter installed, for example, the ChargeCard will take the A000h to B000h block and map memory to it that is contiguous to the traditional 640K, making 704K available to DOS in one nice big chunk—that in addition to being able to map memory to any other unused space between 640 and 1024K.

At one point, ALL boasted being able to make as much as 960K of contiguous memory available to DOS on some machines. This, however, required relocating the video and revectoring calls to video addresses. While that can be done, it also created some problems, so ALL has now backed off from supporting that.

However, it is interesting to note, as pointed out earlier, that there is often at least 16K of address space in the ROM region that can have memory mapped into it for loading a device driver or a TSR or two. The ChargeCard can use this space, but the kicker is that you have got to find it. Typically, this is from F400h to F800h, but don't depend on it. Try it if you like (use a temporary CONFIG.SYS on a bootable floppy while you're experimenting). If the system crashes, try another block. Keep trying until (hopefully) you find an area with stuff the system won't miss if you map over it.

Going this route, the upgraded unit is still a 16-bits-only machine and cannot run 32-bit DOS-extended software. Users are bound to wonder why they would bother now when, with the ALL 386SX, they can go all the way.

About a $200 difference is one real good reason; however, beyond that, a 386SX is not really all that hot. Typically, an 80286 actually will outperform a 386SX (including the ALL 386SX), assuming that they have equal clock speeds.

One of the big advantages that the ALL 386SX and similar upgrade packages enjoy over accelerator cards is that the microprocessor chip accesses whatever memory is installed on the motherboard the same as it always did. All that changes is what you can do with it.

Because of differences in circuit board layout and the location of other adjacent components, it is not always possible to locate the ChargeCard right on top of the original socket. Also, there are several different types of socketing (PGA and PLCC) for 80286s. So, there actually are several different ChargeCard kits, including a slightly more expensive universal kit.

Other than the inherent speed limitation of your existing system, there are some other limitations that should be noted. The maximum memory that the ChargeCard can manage in an 80286 system is 16 Mb, not the 32 Mb normally associated with expanded memory. This is because the 80286 can handle only up to 16 Mb via the extended memory route. You really are dealing with extended memory even though it allows you to use it as expanded.

Like any add-on/add-in hardware, the ChargeCard is a device. Like any device, it requires a device driver not only to identify it so the system recognizes it but also provide the necessary software interface. The ChargeCard comes complete with a proprietary package that centers on ALLEMM4.SYS, which now includes automatic system optimizing. This is similar in many ways to ALL's CHARGE 386 memory manager for 386 and higher systems. It has the same look and feel, but it is different.

Together with its auxiliary programs, the ChargeCard's ALLEMM4 package does a pretty decent job of mapping memory to unused address spaces above 640K. It features an excellent memory mapping utility called ALLMENU (it is one of the best) that graphically displays what is and isn't happening from 0h right to the top—not just to the top of DOS's megabyte but into extended memory, if any is there.

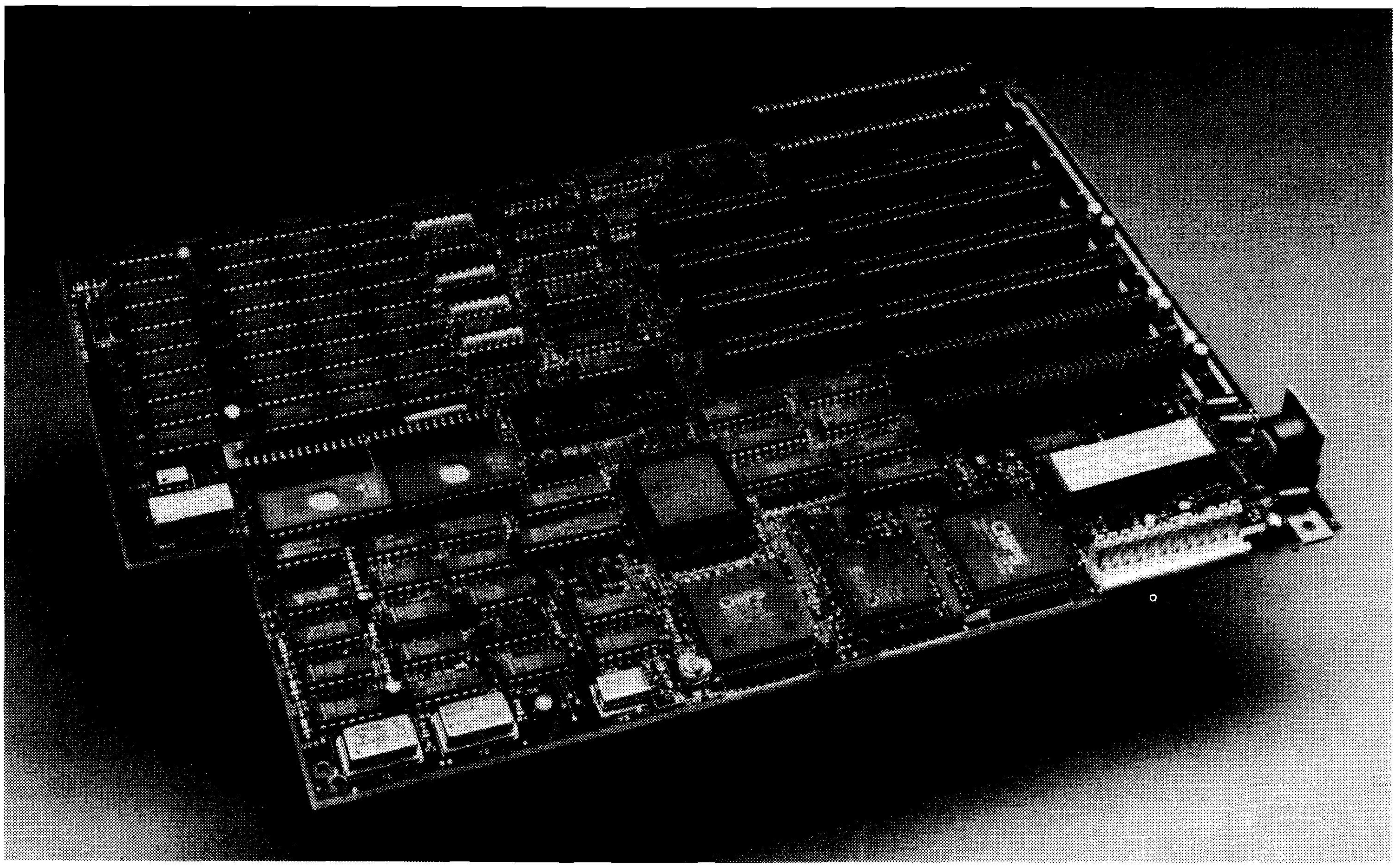
Adopting a new mother

These days the cost of replacing the whole motherboard often costs little if any more than simply adding a typical accelerator card to the old system—even ignoring the no-name mail-order bargains. I do not mean to speak disparagingly of these sources. I've bought my share of closeouts and other bargain boards; however, for motherboard, I'll pay the extra.

Starting from an old 8088, any upgrade motherboard at all—286 or 386—also will move you up to a full 16-bit AT-type (ISA). If you're going full 386 (DX), the better upgrade boards also usually make some provision for direct 32-bit memory access via some kind of proprietary board as well. There should be, but don't just take it for granted, even if there is a 32-bit socket—especially with off-brand boards. While providing a 32-bit socket, some manufacturers of complete computer systems even have been slow to make matching memory boards available for them.

If you really want to go first class, board makers like Hauppauge even offer such optional extras as an EISA bus on 80386 and *i*860 boards and a variety of coprocessor socketing options, even including the *i*860. They provide more choices and a wider range of prices and performance levels from a single source than most manufacturers of complete off-the-shelf machines, most of them more in a range of prices I would consider geared more to new custom installations than souping up that old PC of yours.

There are plenty of those available too, including a number of 80286 (Fig. 16-2) boards. Familiar names, like the once popular AST, XFMR, are no longer on the market; however, they're out there. Some are offered on the mail-order market at prices currently starting under $200, if you really want to squeeze the nickel and you want to chance it. Given all the benefits of going to a 386 and the continuing proliferation of the 386-specific (or higher) software, however, it is increasingly difficult to justify the purchase of new 286 board anymore.



16-2 Replacement motherboards are an increasingly attractive upgrade option and increasingly attractive as a way of creating custom-built machines tailored to your specific needs.

The focus is on the 386 family with SX boards starting at under $400. However, at that price level, you generally are dealing with boards of uncertain origin. In many cases, technical support might range from nebulous to nonexistent, particularly down the line a year or two.

One of the most successful manufacturers in this area—and certainly one of the best known—is Hauppauge Computer Works just outside of New York City. Specializing only in 386 and higher products, Hauppauge has a board for almost everyone, including special motherboards for several specific machines. For example, Hauppauge makes one 386 board that's a special favorite with rebuilders who buy up old Compaq portables just for their nearly indestructible cases.

The steady decline in the price of most new computers puts the ultimate transplant upgrade into a different light, however, if dollars and cents are your primary concern. Typically, for about $2000, you can buy a whole brand new machine these days: new hard disk (80 Mb is almost standard), high density floppy drives, a fresh power supply, the works.

Hauppauge boards start at somewhere under $1000. Anyone who's been inside a PC a few times should have little trouble with the installation (unless you've got an old Compaq or something like my old Colby that's even tighter inside). That is only about half the cost of a comparable new machine and, on that basis, the saving is substantial. However, a penny saved is not necessarily a penny earned in this business.

If it's an AT-class machine you're upgrading—one with a big enough power supply, a reasonably fast access hard disk and a decent high density floppy drive or two—the end result should be a first class performer at a genuine saving. If you're starting with an old 8088, however, you might want to take a closer look before you leap. 8088 machines have slower—and most often smaller—hard disks. Additionally, if it's an older machine, you already might have squeezed most of the good out of that hard disk. (I figure I've got somewhere between twelve and fifteen thousand hours on one old timer in my office now and I'd hate to bet on too much more.) OEMs buy hard disks by the hundreds and they buy them cheap (individually by mail order, figure $400 to $600 for a new one geared to 386 performance). Floppy drives are under $100 each by mail order. A bigger power supply is about the same. Your savings are shrinking fast.

Actually, as a practical matter, getting a nice case is one of the best reasons I can see for gutting an old 8088 and doing a total brain transplant. Old Compaq shells are much in demand, with good 386 reincarnations still bringing top dollar—and not without good reason. With its much better screen and full-size keyboard, I'll take my luggable in preference to my laptop as long as I don't have to actually lug it very far or very often.

Short of custom installations, there are a number of situations that come to mind where an old shell wrapped around a replacement motherboard makes sense. There are networking situations where the processing power of the 386 is all that really matters—so what if the local disks aren't all that great.

When installing any motherboard, at one stage, you will have a lot of connectors hanging loose—connectors from the power supply to the board, from the power supply to the drives you probably had to remove, from the controller cards to the various drives. You want to mark things so that you'll know what goes where and what is right side up.

Not necessarily all of the connectors will be keyed. It's a scary thing to find one that will go on either way. Even if that happens, you're probably not in major trouble. Obviously, things won't work right if you get something plugged together backward, but experience shows that, in general, things aren't likely to start blowing up. They just don't work. So, you turn the plugs around and try again. This way is a little sloppy, but with any care, it shouldn't come to that. Things are rarely as bad as you might think, as long as you watch out for such things as proper physical clearance between things that should have clearance and make sure that all connections are tight.

However, there is another issue that should be addressed at this point, because mating old hard drives with fast new boards involves more than simply bolting all the pieces back together and matching up the plugs.

The interleave factor

One of the lesser understood issues when dealing with hard disks is something called the interleave factor—something made necessary because even slow hard disks usually are too fast for computers to be able to read consecutive sectors. Something has to be done to slow them down to give computers time to digest read/write data before giving them the next sector. Unfortunately, you really can't slow a hard disk down and have it run efficiently, so it's done with something called interleaving.

As the name suggests, what you have to do is break up sectors of data the system must read consecutively by interleaving other data sectors in between them—one, two, or however many might be needed to give the system time to swallow before choking it with more. Obviously, you can't actually interleave the data on a disk. The trick is in the way it's written—writing one, then skipping one, writing one, etc. Reads, then, must be done the same: reading, skipping, reading, etc. It is basically mechanical wait state.

If, by any chance, the system hasn't finished swallowing the last byte (pardon the pun) by the time the next sector it's supposed to read comes along, it skips on past and has to wait until the drive does another full revolution for that sector to come back under the head again. That wastes time—a whole extra turn. The more sectors that have to be read, the more time wasted on extra turns. Conversely, if the interleave is set too high and always makes the hard disk twiddle its heads while one or two or several extra segments pass, that wastes time, too—not as much as skipping a turn, but enough to degrade performance.

The thing that controls the interleave factor generally is established when you first set up your hard disk. Once it is established, your hard disk is formatted on that basis. Typical interleaves range from about 4:1 for 4.77 MHz 8088s to 2:1 for 8 to

10 MHz machines and sometimes 1:1 with 80386s. A number of factors enter into it, but clock speed plays a major role. So, clearly when you install a device—a system board or accelerator card with a faster clock—the timing changes. So, if the interleave was right before, it won't be when you're finished. (See Table 16-1.)

Fortunately, interleave factors can be changed. To make the most effective use of an accelerator board—especially in an 8088 machine—it's one of the first things you should look into.

Table 16-1 Interleave factors and effective transfer rates. This test was conducted on a PC upgraded with an Intel Inboard386-PC. However, this 16 MHz upgrade does not result in a bus speed increase that would support a tighter interleave than the original interleave of 5. A factor of 1 or 2 would be characteristic of a new system designed around an 80386. (The test data was obtained using Paul Mace's HOPTIMUM software.)

Interleave	Effective transfer rate	Time
1	28.3	75.0
2	26.8	29.1
3	25.5	83.3
4	26.8	79.2
5	85.0	25.0*
6	85.0	25.0
7	72.7	29.2
8	63.7	33.3
9	56.6	37.5
10	51.0	41.7
11	46.4	45.8
12	42.5	50.0
13	39.2	54.2
14	36.4	58.3
15	34.0	62.5
16	31.9	66.7

*Relative Best

Special software (HOPTIMUM) for checking the efficiency of the interleave factor and changing it if indicated is available from Paul Mace. While HOPTIMUM is well-written, changing the interleave factor is a tricky process that, when carried to conclusion, must reformat your hard disk sector by sector. Data from the sectors that are to be reformatted is read into memory, then written back. Generally, it comes off without a hitch, but you most definitely want to back up anything you can't afford to lose before you start.

You can buy a faster hard disk, but then you're running up the cost of the upgrade by perhaps half the cost of the accelerator card, which is likely to make you want to think twice. Even then, just because a certain hard disk is capable of delivering a certain level of performance is no guarantee that that's what you'll get. HOPTIMUM is still a worthwhile bet, if only to satisfy yourself that your system really is performing at capacity.

Superfast "disks" and disk caching

Most of you probably have used RAM disks at some time. Your AUTOEXEC can set up one routinely every time you boot to provide real fast disk access for temporary files. At very best, a mechanical disk of any kind is going to be a laggard by comparison.

Today, as bigger databases, bigger files, and bigger programs appear, disk I/O becomes more of a major bottleneck. You don't have to wait for huge DOS-extended applications to know that. If you work with dBASE very much, you know how much time you can lose there even now on disk I/O. You can use disk caching to move things in and out a little more efficiently; however, you're limited by how much precious memory you can spare for caching. Even 1 Mb is only a drop in the bucket compared to the aggregate size of files and data for many applications.

If you've got the memory to spare, you can make the cache bigger if you need to. If your needs for a larger cache are more occasional, this is probably the better way to go, leaving that RAM available for other uses other times. Where the need is fairly constant, you might do well to consider dedicated disk caching hardware. A line of AT-bus boards that are fairly typical of this type of cache, with capacities to about 10 Mb are available from a firm called Distributed Processing Technology.

You also can use a RAM disk—no ordinary RAM disk, mind you, but megabytes of RAM disk: ten, a hundred, a thousand, even more. Imagine a RAM disk big enough to download your entire hard disk and having RAM-speed access to everything. You won't find them at your neighborhood computer store probably, but they are available.

A major supplier of modular expandable RAM disks is Newer Technology, a firm mentioned earlier in conjunction with fast memory cards that could operate dependably to bus speeds of 14 MHz. Its DartCard, which fittingly mounts in a full- or half-height drive bay (depending on the model), can be user upgraded in 4, 16, or 64 Mb increments. If 4 Mb chips are used, a single DartCard full-height assembly can be expanded to as much as 704 Mb and multiple units can be daisy chained.

While marketed primarily as a RAM disk system, RAM is memory. The memory on a DartCard can be accessed as extended memory, if desired. Up to 8 Mb can be accessed as LIM 3.2 EMS expanded memory. (There were no plans at this writing to upgrade this device to fully implement LIM 4.0 EMS features.) It is available with optional interfaces including PC (8-bit), AT, SCSI, and ESDI and optionally can be powered continuously and configured as a nonvolatile, BIOS-compatible bootable disk.

As you can see, there are several valid options to consider before just throwing in the towel on what you've got and going out and buying something new. Upgrading your existing unit might give you all of the computing power you need now and for some time to come.

It is a paradox that IBM, the computer that developed that curious toy called the PC—the PC nobody quite knew what to do with when it first came out—gave us open architecture. With that open architecture now, the means are available to make even that old PC act young again—like a fountain of youth. You can be sure someone didn't think that one through.

17
CHAPTER

The ultimate upgrade

At some point, none of the easy remedies (adding an accelerator card or another memory card) are going to be enough, particularly now, with the spotlight focused on extended, not expanded, memory. You might be at that point already—the point of trading that old box of yours in on a newer model or even better, relegating it to someone you don't like in another office.

Where do you go from here, however? Given the options and the changing focus of the industry, is an 80286 enough? Is it worth the little extra money for an 80386SX? Should you go the extra distance for a full-blown 80386DX? Then, there's the *i*486—the chip someone affectionately called the "the Grinch that stole the coffee break" because users never have to wait for it. If you're heavy into number crunching maybe a 486, with the equivalent of a million transistors, is the way you ought to go. It's generally three to four times faster than a 386 at the same clock speeds but still a full-fledged backward-compatible member of the 8086 family.

If that still isn't hot enough to suit you, they now are teaming *i*860 chips with *i*486s on some machines to boost number crunching speeds still higher—four, five, or even ten times faster. Talk about blowing the doors off. This can be done only with special software, because the *i*860 is a RISC (Reduced Instruction Set Chip) and, as such, is not compatible with ordinary DOS-based software. It can be and is being done.

Coming back down to earth though, aside from crunching numbers faster (due in large part to the inclusion of what amounts to a math coprocessor in its design), in the long term, an *i*486 can do nothing that an 80386 can't handle. It is still just a 32-bit chip, no more. Unlike the 386, it will not spawn a whole new genre of software.

It appears that the 80386 should remain the darling of the PC set for some time yet to come. Few users that have 386 have pushed them to their limits, nor are we likely to real soon. The issues, however, are not as cut-and-dried as they might appear.

When the *i*486 is discussed in terms of number crunching, the first image that is conjured in many minds is heavy scientific applications. In general, however, CAD, desktop publishing, and graphics packages are essentially just number crunchers. Just generating the screen displays requires a lot of number crunching, as evidenced by the fact that the display adapters for some resolution systems now mount up to 4 Mb of video RAM. A 486 might well be the most cost effective machine available today for many such applications.

While on the subject of graphics, what about Windows? Not only a real memory grabber like any bit-mapping program, Windows—and Windows-oriented applications—impose a heavy overhead on the CPU. As a rule of thumb I'd say anyone contemplating going from a character-based to a graphical environment should probably figure moving one step higher on the CPU ladder.

For other applications, however, the 486 now is seen as having too much power. Originally promoted as being the ideal heavy-duty network server, its power now is seen by many as overkill in this arena. You can only push data through a network so fast.

Striking a happy medium

At this point, few people could disagree that the 8088 has, to a great extent, outlived its usefulness, except in rather modest circumstances. Do you need a 386 or will a 286 do just as well and cost you less?

I have shown that there are ways to get around the 286's problems getting from protected mode back into real mode and back into DOS. With the kind of added hardware help the All ChargeCard offers 286 owners, the practical differences between a 286 and a 386 can be narrowed greatly, disappearing altogether for many applications when measured against a 386SX.

As a purely practical matter, how many of you will ever want or need to go beyond the 286's 16 Mb extended memory limitation? However, that really is not the question. The question increasingly is one of 16-bit versus 32-bit systems and 16-bit versus 32-bit software. Even software that started out as 8-bit packages is leap-frogging right on past the 16-bit 286 market to take advantage of 32-bit processing speeds, thanks to some of the new extended memory development tools.

Possibly the best advice to anyone considering the purchase of a new 286 at this point would be not to if you can avoid it. The problems inherent in the 286's design are not going to go away. As extended memory becomes more important and DOS-extended 32-bit software becomes more common (which won't run on a 286), the validity of the 286 machine comes more into question, particularly because the price differential between an SX and comparable 286 has slipped at this point to something typically around $200.

You can buy the 286 now, knowing you can upgrade with a ChargeCard later if you like. With a ChargeCard, you can even run a lot of 386-specific software—but not all and don't let anyone try to tell you otherwise. At today's prices, a ChargeCard upgrade will cost you roughly twice any "savings" that you hope to realize up front.

Given the competitive price and the benefits of full 32-bit processing power and most of the other more important features of a full-blown 386, the SX is seen by many as replacing the 286 altogether in mid-priced machines in the not too distant future. However, I think it's too early yet to completely write the 286 off. Still, in bridging the gap between the 8088 and 80386DX, the SX rather neatly fills shoes that never quite fit the 286.

The 386SX, from the beginning, supported clock speeds of at least 16 MHz. Those speeds, not surprisingly, have gone higher. You have to be careful of clock speeds, however, because it's too easy for vendors to slip in wait states that slow the operating speed down to accommodate system board designs and cheaper components that might not support the blistering speeds advertised.

Another vendor trick I have discovered is using clocks with variable speeds, so that even if a machine boots at 16 MHz, you might find it running at some far lesser speed once you put it to work. In a classic case in point, I measured the net effective speed of one 16 MHz 386SX at only 1.3 times the speed of an old 4.77 MHz 8088. The manufacturer did not see fit to address this situation; therefore, I must conclude that this is not abnormal for that machine. Unfortunately, the unwary buyer, lacking the tools to benchmark machine performance, might never know the difference.

Similar tricks can be used in designing around 486s, as well. Just putting in a fancy chip does not guarantee performance, as I'll show you a little later in this chapter. As clock speeds keep going higher and 486 prices are becoming increasingly attractive, the wary buyer is going to have to be more and more careful.

Lest there be any doubt, however, the SX is not just a marketing gimmick to allow Intel (and O.E.M.s) a two-layer price structure for the increasingly popular 80386. It is a different chip that, while enjoying most of the operational features, is also quite different in other ways. For example, the maximum physical memory (RAM) that the 386SX will support is limited to 16 Mb—the same as the 80286—rather than the 4 gigabytes of a full-blown 80386. (The 386SX has only 24 address pins—the same as the 286)—hence the similar address capability.) However, it can be used with up to 64 terabytes of virtual memory and supports a maximum segment size of 4 gigabytes. (Those numbers are identical to the original 80386.)

What the SX does not support is a 32-bit bus architecture. Like the amount of physical memory addressed, this limitation is a function of the number of pins the SX chip has—32 less than its big brother. (The 80386SX chip is a 100-pin "Quad Flatback" device; the regular 80386 has 132 pins and is housed in a much larger "Grid Array" package.) A 16-bit bus, however, does not limit its processing power, only its bus access. The reduced complexity of a 16-bit versus a 32-bit bus, combined with other factors, makes it possible to significantly reduce manufacturing costs for systems boards that support the SX chip.

As a DOS machine, the SX retains the full address remapping capability seen first in the original 386s. There is a companion 80387SX coprocessor available for heavy-duty number crunching. Certainly for most users (and most applications), the difference between full 386 and 386SX machines should be negligible—if they are noticeable at all.

Still, make no mistake; the SX is not a high-end machine at a mid-range price. Even with desktop performance that was unimaginable just a few years back, many people already look upon the SX as an entry-level machine. When you consider that the price of a typical SX today is less than you would have paid for a 64K 8088 PC with a green screen not that many years ago, the SX really is an entry-level machine.

As the price of the SX has slipped, so have other prices, to a point where *i*486s are selling at the prices of last year's 386DXs. Hauppauge, one of the top system board manufacturers, currently even includes a free *i*860 coprocessor and special software to drive it, as an added inducement to buy its top-of-the-line 486 board.

Cash and carry

One of the most phenomenal growth areas in the industry is the laptop market. Developers have now downsized them to notebook sizes, weighing only six or seven pounds. In many cases notebook computers pack the power of a 386 and are upgradable to support 8 Mb or more of RAM. In some cases, they even support math coprocessors, as well.

Prices generally are significantly higher for these mini-wonders than for their desktop counterparts, which is due in a large part, to the significantly higher cost of the flat screen technology laptops require. However, with increasing competition, even in the laptop/notebook market, prices have been dropping almost as fast as high-end performance has increased, making them an increasingly attractive alternative.

Limited battery power, often restricting cordless use to a couple of hours at best, continues to be a problem, as with any portable device. You should not take manufacturer's claims too seriously in this department. I have found in some worst-case situations that advertised operating times could be attained only with the screen turned off more than 90% of the time and little or no floppy disk activity, even on machines that only have floppy drives. Still, as a practical matter, a couple of hours of serious computing is about all I have time for on most cross-country flights, which is about the only time some alternate power source is not available.

Purchasers anticipating international travel should be sure the charger/AC power supply supports 90 to roughly 220 volt 50/60 cycle operation (carry the appropriate adapters). Also, some countries, particularly some of emerging nations, have peculiar regulations regarding the "import" of any high tech devices. In some cases, they do not allow their entry; in others, customs officials note their serial numbers next to the immigration stamp in your passport to be sure that the same devices leave with you. These regulations also are subject to change without notice, particularly in times of international tension when electronic devices are looked on with particular suspicion for security reasons. The best advice is to check with consulates before you leave home, if there is any question, as well as with whatever airline you are traveling.

Down memory lane

The transition from individual chips to SIMMs probably went unnoticed by most users. Initially, they seemed to make little difference. Granted, a 256K SIMM took up only roughly half the space of nine individual chips and was a lot easier to plug in; however, in practical terms, it made little difference. The speed of the chips was what mattered most, no matter how you packaged them.

Those days are gone, however. Today, 16 Mb SIMMs exist, packaged on a single little plug-in module requiring no more than one quarter of a megabyte. A motherboard with socketing for four SIMMs that once might have added up to just 1 Mb (at 256K each) now can hold 64 Mb. Someday, 64 Mb SIMMs could raise up the ante to 256 Mb—a quarter of a gigabyte right on the system board.

Admittedly, even with the 80386 DX and *i*486, such numbers have no relevance in today's DOS-based world. Given the explosion in memory usage over the past decade as a baseline, it is not hard to project a time in the not too distant future when those numbers will have relevance. Unfortunately, when that day arrives, the socketing that you have might not be ready for them.

The actual socketing for 64 Mb SIMMs would have to be identical to the socketing for the more mundane denominations present today. Few, if any, actually would. The exact number enabled by different manufacturers is largely a matter of economics weighed against what individual vendors perceive as necessary in some cases, or merely expedient in others.

To put this in perspective, in most cases, there would be little point in having all these lines enabled. No matter what size chips a SIMM is made of, the number of kilobytes or megabytes than can be addressed is still a function of the number of data lines, whether they are pins on the processor chip (as discussed back in chapter 1) or enabled data lines—lines that actually are connected to the CPU.

A good question for the prudent buyer to ask in today's market is what is the maximum size SIMM that the device can accommodate. Don't just take "Oh, any size I guess" for an answer.

Ladies-in-waiting

Intel has demonstrated a CPU chip running at a blistering 100 MHz. As I stated in the last chapter, however, there are technical limitations that make it impossible for many devices to run even at today's more mundane clock speeds. To get around these problems manufacturers inserted wait states that effectively slowed the system down to operating speeds that are much less than the clock speeds. In the last chapter, I was talking about the problem as it related mainly to upgrading old machines, but it is not limited to older machines by any means.

RAM chips can handle data only up to some finite speed. This is a fact of life, as is the fact that different types of RAM chips have greatly different speed potentials. Clock speeds have long since been reached that are well in excess of what DRAM—Dynamic RAM—chips can keep pace. To an extent, that limitation can be masked by interleaving banks of RAM, alternately addressing one bank and then another. There are practical limits to how far any DRAM scheme can go effectively.

SRAM is much faster but also is more expensive, taking away much of the edge cheap memory has brought. There are ways the two can be mixed, however. ComputerAdd has been a leader in this area, with a top-of-the-line tower that, although using mainly 80 nanosecond DRAM, will support an optional daughter board with enough 25 nanosecond SRAM to run the DOS conventional memory area with no wait states. This technique is viable up to about 50 MHz.

To go much farther—to even approach 100 MHz—something else will be needed, however, because SRAM does not lend itself to interleaving, not that users have to worry about 100 MHz systems for a while yet anyway. The point is that you should not be fooled by clock speed claims alone. I have in house, for instance, an AST 16 MHz 386 SX that consistently benchmarks 15 to 20% slower than a 12 MHz 286 machine that my wife uses. If you buy them that uses wait states, you probably are not getting the performance you thought you paid for.

Don't miss the bus

As CPU speeds have climbed higher and higher, bus speed and design also have become increasingly more than just critical issues with desktop machines. They are the limiting factors in many instances. I addressed this issue when I discussed expanded memory boards in an earlier chapter, in which I discussed upgrading the system you have now on your desk—likely something with the standard AT-type 16-bit bus that still is so common.

Unfortunately, the PC/AT bus, which was adequate in its day (although hardly inspired by genius except for the open architecture concept), has been made obsolete, to a considerable extent, by today's technology. Even now, many memory boards have difficulty coping with today's bus speeds in some machines. The problem isn't simply one of using faster chips. There are problems inherent in the basic structure of the bus itself. These problems, it seems, can be addressed only by completely changing conventional thinking about bus design.

It's been done. That's what's behind the Micro Channel Architecture (MCA) that IBM introduced with the PS/2. Unlike the old PC/AT bus, MCA has not become an instant industry standard. For one thing, IBM was careful to tie the Microchannel up in all kinds of patents to assure that no one would use it or design third-party products without IBM's blessing (and license). So far, the company has been quite selective in licensing the use of those patents and the indications are that it will continue to be so.

I'm not saying that IBM is playing dog-in-the-manger with the MCA; however, I'm also not saying that there might not be at least an element of that, either. After all, the open architecture of the old bus clearly got away from IBM when suddenly the foolish toy it'd created turned into serious business—for almost everybody else. Don't kid yourself; IBM is out for one thing: IBM.

Still, some of the top third-party expansion product manufacturers were in there quickly, almost from the start, with add-ins of various types. Between licensing and other factors (including special VLSI chips needed to implement some of these features) do not expect a glut of Microchannel products, as happened with the old style bus.

IBM, however, was not the only developer involved in developing a better bus. The result was that several other bus designs were suggested. One design in particular has attracted considerable notice and already is being implemented in top-of-the-line computers from several well-known manufacturers. Called the EISA bus (Extended Industry Standard Architecture), it developed out of the collective thinking of Compaq, Zenith, AST Research, Wyse, Hewlett-Packard, Olivetti, NEC, Tandy, and Epson.

Philosophically, there are many similarities between these two leading contenders, reflecting a general consensus on the part of participants in both camps with respect to areas of greatest concern. However, there also are significant differences. Because the purchase of any new computer represents a substantial commitment to a particular bus technology, it is something that should be weighed carefully when buying new equipment.

What's wrong with the old bus?

A lot of things are wrong with the old bus. Probably the best way to understand what's so good about what's new is to look at what's so bad about the old machines. I cited today's sizzling clock speeds earlier. That actually is one of the lesser considerations.

One of the biggest problems with the AT bus is that no card can access more than just a single interrupt line. What that means is that a computer with an AT bus, in effect, has a one-track mind. If you issue a disk read/write command, the machine cannot write a screen update until it finishes with the disk I/O. In the meantime, if you need to print a file, that file has to wait its turn—i.e., for whatever task is using the interrupt line to release it.

Further adding to the jam-up, only the CPU or DMA (Direct Memory Access) controller can take control of the AT bus, one job at a time. Users pay a healthy premium to buy fast access hard disks, then keep the disks waiting for the bus. Nothing can run a peak efficiency: not the CPU (which is continually being distracted with housekeeping details) or any other resources that must stand idle much of the time waiting their turn.

The old PC/AT bus is, at best, a 16-bit bus, even on machines with 32-bit CPUs, like the 386. God forbid you give a 486 an AT bus. The way many 386s have weaseled around the 16-bit problem to date—and low- to mid-range units will no doubt continue to—has been to stick one proprietary 32-bit socket on the motherboard. The socket that they use is one that generally is usable only for additional RAM installed via some kind of proprietary (non-standard) 32-bit expansion board. Everything else is just plain old 8/16 bit stuff.

Speed is a consideration; however, the fact that, regardless of clock speed, AT bus speeds rarely exceed 8, 10, or occasionally 12 MHz has more to do with the fact that most of the adapter cards (video, expansion, etc.) designed for use on AT-bus machines cannot deal with the higher bus speeds that today's higher clock speeds deserve. So developers slow down the bus and throw in wait states where they have to, all in the name of compatibility with the hundreds of thousands of existing AT-bus adapter boards. This is a real waste.

Beyond "advanced technology"

IBM's Microchannel not only can but does run at sizzling bus speeds—CPU speeds, with no delays and no waiting. And it is a true 32-bit bus all the way. This is especially important with 386 and *i*486 processors that are 32-bit chips, because it means that any adapter board plugged into any open MCA socket can be a full 32-bit board.

One of Microchannel's most significant features is advanced bus arbitration. Translated into performance expressed in ordinary English that means not only faster data transfer rates but—and this is the biggie—several signals can share the bus simultaneously. The conventional PC bus can handle only one operation at a time (move a block of data to or from the hard disk, for instance). The Microchannel can do that and handle seven other data transfers at the same time. It is a multitasking bus, if you will.

Underlying this multitasking ability is a change in the entire DMA (Direct Memory Access) design philosophy. With Micro Channel Architecture, DMA channels—eight of them—move data without having to use the microprocessor to manage the logistics. The DMA is controlled by a separate chip, which is really little less than another microprocessor, a coprocessor, that's been specially designed just for the job of traffic management.

With the Microchannel, the host computer only has to tell the DMA controller chip what, where, and when, and then go on about its business computing while the DMA controller does its thing. In contrast to the traditional PC bus, when a single DMA transfer takes place, almost everything else must come to a screeching halt.

The Microchannel is a smart architecture, too. You can forget those pesky DIP switch settings when changing your configuration. With the Microchannel, it's all done automatically using software. In case of a fault (a failed memory bank, for instance), a properly designed MCA board can quietly bypass it and go on without it.

What might not prove so smart is buying into MCA if you have invested heavily in AT bus adapter boards. There is no practical way of adapting them. When you commit to MCA, you are committed.

When the world outgrew the 8-bit bus, the 16-bit AT bus architecture that succeeded it did not obsolete it in the way the 32-bit Microchannel obsoletes the 8/16-bit PC/AT bus. When the AT came along, the bus structure it offered would not only accept a whole new generation of advanced 16-bit add-in cards, but also most of the older 8-bit cards, as well.

The importance of this compatibility is even more significant today because even as users move more and more toward 32-bit processing and beyond, there is no way for many of the boards used today to benefit from—or even use—a 16-bit data path. For instance, I/O boards for serial and parallel ports can only use an 8-bit data path, as can modems or interfaces for special pointing devices, etc.

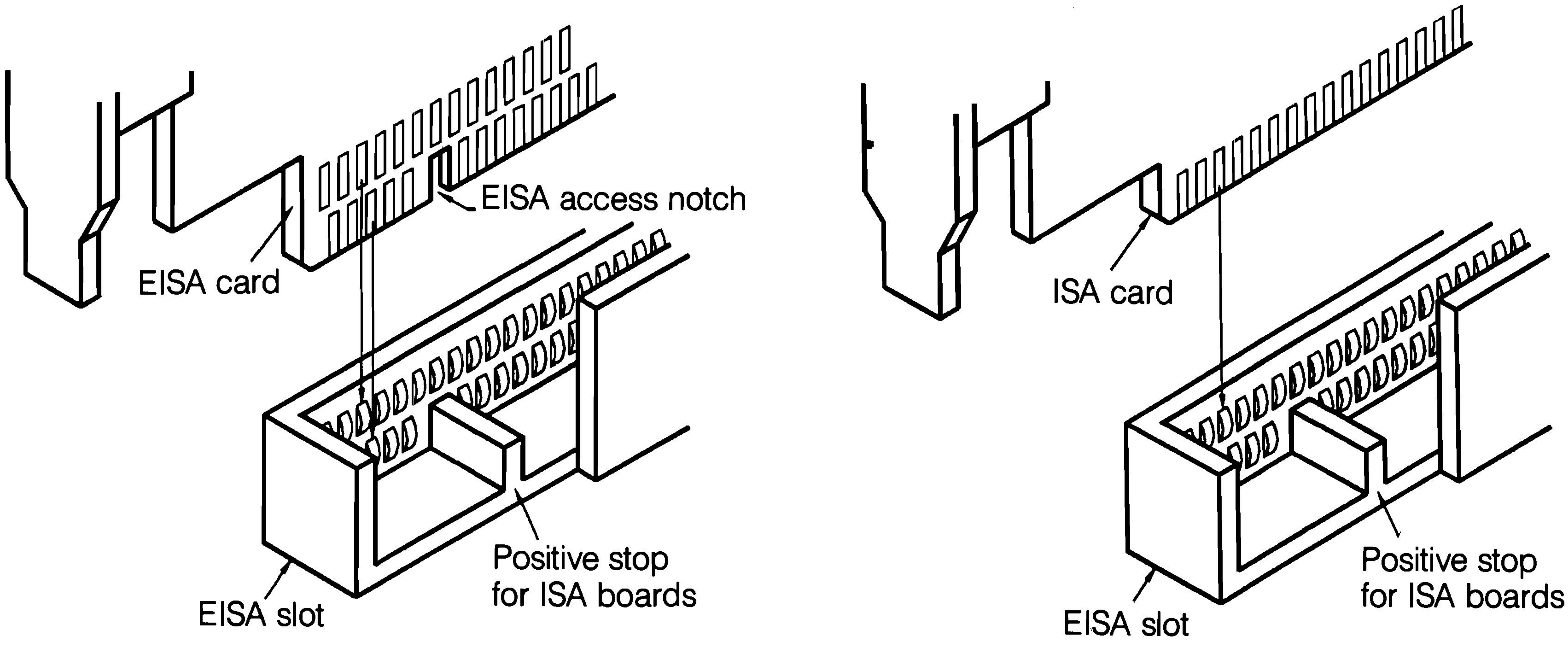
There is currently a number of 16-bit video adapters. With the exception of adapters for some of the higher resolution displays, however, you generally have a choice it seems, making it possible to retrofit new monitors to systems having only 8-bit buses.

Now, you can mount any of these on fancy 32-bit boards. Unless you have to—unless the application needs the wider data path—there's nothing to be gained. You will probably pay a higher price tag to cover the cost of repackaging.

This is where the EISA bus, which is descended from something called the ISA (Industry Standard Architecture), comes in. The ISA bus is the comfortable old 16-bit AT bus, given a formal specification and a fancy name.

EISA is an Extended ISA bus. That's more than just a fancy name. It is a very clever answer to the thorny problem of maintaining compatibility with the myriad ISA (AT bus) adapter boards out there.

Remarkably simple in concept, the EISA bus is designed around the use of a bi-level socket that is an ISA bus socket on top. Under that, however, there is a second set of contacts (slightly offset) that an ordinary ISA board can't reach as shown, in Fig. 17-1. The result is a bus that doubles as a true 32-bit bus for EISA adapter boards or a 16-bit bus for ISA boards. At least one manufacturer (ALR) has already pushed beyond this basic 32-bit data path extension to create a 128-bit data path device that is still fully compatible with the standard 8/16-bit AT bus structure.



17-1 The left side of the figure is a cutaway representation of the EISA bus. The most important and distinguishing feature here is the second (lower) set of contacts, both in the socket and on the card being inserted, making data paths as wide as 128 bits and making other exotic features practical, while retaining compatibility with ISA boards, which is one of the big selling points for EISA over IBM's Microchannel. In the right side of the figure, you can see how the ISA card, with a shorter and unslotted skirt, is prevented from reaching lower contacts by positive stop.

With proper engineering, the EISA bus can provide most, if not all, of the advantages touted for IBM's Microchannel—plus a few of the MCA can't match. Development costs can be significantly lower for EISA boards in many cases. With nearly twice the surface area of MCA boards, EISA boards often can be designed around more conventional components that do not require the more expensive surface mount technology widely employed on MCA boards.

At least one manufacturer (AST), carrying the EISA bus philosophy one step farther, has designed its top-of-the-line computers around interchangeable ISA and EISA system boards. Breaking with traditional design concepts, the ISA and EISA boards are little more than glorified bus boards. The CPU chip and logic circuitry are functions of separate plug-in boards that can be mixed and matched with system boards that hold either ISA or EISA buses. That scheme allows for incremental upgrades of the bus, the CPU, or both, as needed to keep pace with changing needs.

ALR, one of the most interesting and innovative players in the game today, also has gone the plug-in CPU board upgradable route; however, ALR adopted its own EISA-compatible bus at a more attractive price. (The ALR bus adds even greater flexibility to the basic EISA-standard bus.) It truly is a buyer's market.

It still is too early to declare a winner in the bus wars and, indeed, there might never be a clear-cut winner. It is interesting to note, however, that, despite IBM's two year head start with the MCA, as of this writing more than 40 manufacturers were either marketing or known to be committed to developing EISA-based computers and/or add-in boards. At the same time, other manufacturers are betting on IBM's track record. Still others are hedging their bets, developing and marketing both MCA and EISA hardware lines.

It is doubtful that users have even glimpsed the best that IBM and ALR have to offer. For instance, there are a lot of other tricks that systems designers can use to increase the effective I/O speed. Several AT-bus machines using data caching actually demonstrably outperformed IBM's early implementations of the Microchannel. By using some of these same—or other—tricks, both camps might well develop still faster buses in the months and years to come.

Putting the pieces together

I've shown you at least a fair cross section of the hardware and the software that is out there. You have seen computers designed with no memory of their very own and, at the other extreme, computers that either own whatever memory they have or wastefully disable it so nobody can have use of it. Most of the industry lies somewhere comfortably between these two extremes. Those companies are too comfortable perhaps, with too many still selling old technology even as the world around them has turned upside down in many ways.

That does not necessarily mean that users should buy only computers that have zero memory. If users study the lessons that computer history should teach them, buying a zero-memory computer might not be a bad idea. Even more than dazzling supersonic clock speeds, memory management is and will continue to be one of the most critical issues. There's an interesting story there, too.

I talked earlier about the origins of EMS; however, I said nothing about the origins of the EEMS that was the foundation stone of expanded memory as it is known today. EEMS, which essentially said that memory should not be owned by the CPU but controlled independently, was a concept—like the original PC itself—that had no practical application. There were some clever people at AST who, motivated by whatever logic there might have been, simply said there had to be a better way.

So, AST created its own standard, which was compatible with the then accepted LIM 3.2 EMS but went an extra distance into uncharted territory. It was not done to meet a need—there was no need. AST simply did it because the company thought that was the way it should be done. It was a better "whatzit" trap, except nobody knew what a "whatzit" was. As long as you could sell it to catch mice, however, it was okay. If nothing else, it was good for bragging rights—sort of like having a star named after you.

Then, a bunch of clever guys came along with DESQview and said, "Wow! We can do multitasking with this." AST had their "whatzit" and the world had multitasking.

The moral to the story is simple. There are few really bright ideas in this business (no matter how disembodied as they might seem at the moment) that cannot find embodiment in the fulfilling of someone else's dream. Even the false starts succeed sometimes (they laughed at Columbus).

Software can always be upgraded if something better comes along—something backward compatible with your hardware. Hardware, however, is quite a different matter. If the hardware was not designed anticipating change, there generally is little you can do about it later.

What does it all mean to you? Obviously, each individual situation is different. From almost any standpoint, however, the 80386 is rapidly emerging as the platform to build on as the computer moves into the 1990s.

This is not to say that everyone should rush right out and buy the full 32 Mb allowable under the 4.0 LIM EMS—or even 8 Mb or 4 Mb. Big blocks of memory, although very cheap today compared to just a few years back, are still expensive relative to the base price of your CPU. As with most things, the trick is to strike a balance that is adequate for current needs or just a little more.

Just an extra megabyte or so (when used with Microsoft's Windows, Softlogic Solution's Carousel, or Quarterdeck's DESQview—three popular windowing environments allowing more than one application to be loaded simultaneously) will probably allow you to load your word processor, spreadsheet, database, and probably a couple of utilities, all at the same time. That depends on just how big each of those applications and their files are. If you're talking average or typical small office applications, however, that might well be enough—at least for now.

The problem is that an extra megabyte or so can open up so many doors that you're likely to need still another megabyte before too long—and then another two perhaps. With the right choice of hardware, adding more memory should be no problem. However, there are some hard choices that should be made, especially with some add-in memory product.

Software will come and go and be replaced by better software. It's inevitable. As pointed out so graphically by the sudden surge of DOS-Extended software, how much of tomorrow's new software that still will be able to run on the hardware you buy today—or how well it runs—will in large measure depend on how carefully you select new hardware now.

Recycled oldies

There really isn't that much market for those old PCs, XTs, or even ATs out there and even less for those old boards you've stuffed in the expansion sockets. The age of a high-tech equivalent of the used car lot has not yet arrived. Many dealers don't even want to take old hardware off your hands.

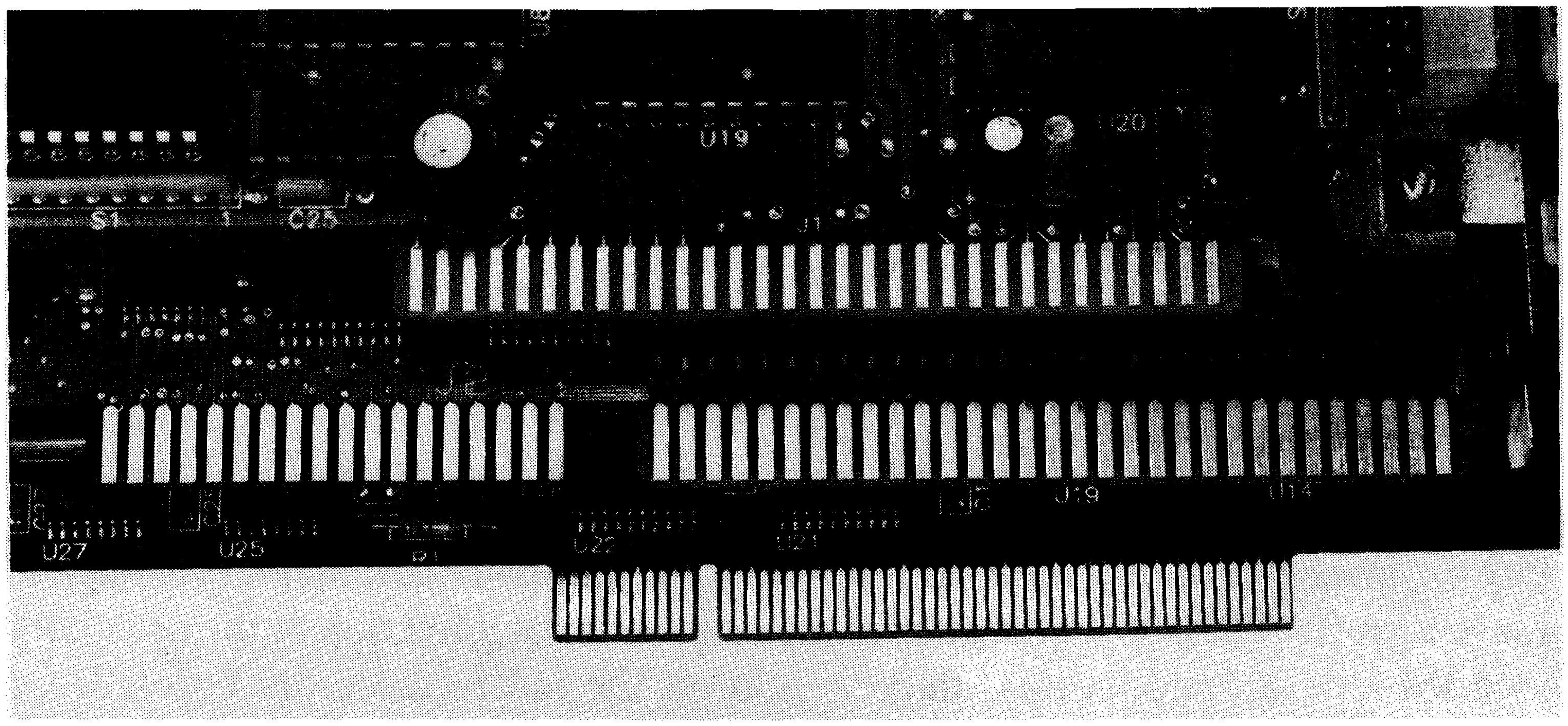
There is some market for used hardware; however, you've got to do a little digging. Some companies specialize in selling used machines, provided they can buy the machines at bargain-basement prices, so they can mark them up enough to make a profit. These companies offer their machines at a fraction of the price they sold for new. In many cases, there is a lot of life left in those old boxes. It's a shame to throw them out—or very nearly so, considering how much you get for them. Today more than ever perhaps, most offices have more employees who could benefit from having a computer than the budget has allowed so far. The idea of just putting one on everyone's desk as a stand-alone unit is obvious and certainly needs no elaboration.

Less obvious, perhaps, is the idea of using the computers as either smart or dumb terminals in one of the increasingly popular and cost-effective multiuser environments. In chapter 10, I'll look at some environments where those old desktops might well save you the expense of fancy workstations. There's multiuser PC-MOS, Concurrent DOS, and DOS add-ons like VM386.

There also is full-scale networking. LAN (Local Area Networking) opens up all kinds of possibilities. Also, autonomous multiuser clusters can be tied together on a network. There are all kinds of ideas. Speaking of ideas, it's not a bad idea to keep an old machine or two around as backups for those times the fancy new ones throw a fit. (I had to drag one out not too long back when one went down.) Don't be too quick to get rid of those old workhorses, even those old 8088s, not until you've carefully checked out all the options and considered all the possibilities.

No no no!

A question that inevitably comes up is about reusing old 8-bit PC expanded memory boards in 286 and 386 systems or reusing even older 16-bit boards in many of today's sizzlers. They will, after all, fit nicely. The 8-bit boards fit into almost anything, and the 16-bit boards fit into any IBM AT-type bus (not Microchannel) 286 or 386, as you can see in Fig. 17-2. They will fit. The likelihood that they'll work is something else. If you were making odds, they'd have to range from slim to none.



17-2 Visual comparison between the 8-bit PC (top), 16-bit AT (center), and Microchannel bus connectors (bottom). 8-bit cards will fit 16-bit socketing. Many systems with 16-bit bus structures commonly use 8-bit video cards, modems, FAX boards, etc. Bus speed considerations, however, generally make it unfeasible to use 8-bit memory cards on a 16-bit—especially considering the problems that some 16-bit cards have keeping up with some of today's sizzling bus speeds. Except for a few computers that offer special bus adapters, there is no interchangeability between Microchannel cards and either of the other two cards.

It's not just manufacturer's hype, trying to sell you something new when something old might serve just as well. Expansion boards (third-party or otherwise) are designed for use in specific types of system: the original 8-bit or 16-bit buses in the PC and AT and more recently, the Micro Channel Architecture in IBM's PS/2 line. The very thing that makes these new machines such sizzlers obsoletes a lot of older memory hardware: speed. Not only have CPU speeds climbed dramatically, but with them, bus speeds as well.

"Sure," you say, "A faster CPU means a faster bus. It has to be." However, it is not quite that simple. Bus speeds rarely are the same as CPU clock speeds on today's computers. There are few buses that exceed 12.5 MHz; most are slower. Some of today's boards even have a problem keeping up and some can't.

Having said that, it should be clear that you should not expect to save a few bucks if you trade your old box on a dazzling new one by reusing some of those old boards—even pretty recent "old" sometimes. You can try them, but at your own risk. Most manufacturers just plain won't support them in this kind of usage.

I'll close this chapter with one final caution. Before you buy a machine that requires proprietary accessories to access any special features of that machine, make sure the accessories actually are available and not just in the planning, prototype, or wishful thinking stage (in the trade, they call it "vaporware") if it's something you know you're going to need. Remember the 386 I cited earlier as having come to market with a proprietary 32-bit slot—as most of them do—but keeping customers waiting for a year or more before there was a proprietary memory board to fill it. Do you remember the PCjr? These things happen to the big guys, too.

18
CHAPTER

Crash course

The word "crash" took on new dimensions when users slipped the bonds of DOS's old 640K. We have come a long way since then. Software engineering has evolved from the realm of wizardry and witchcraft into a science, with the result that today's software is far more robust and reliable.

However, at the same time, with increasing competition for whatever address space that can be scavenged above 640K, users face another set of problems. It was the LIM EMS and EEMS specifications that first made legitimate use of address space above 640K and brought to everyone's attention an area that IBM and Microsoft had posted big "Reserved" signs on.

Things had been going on up there for a long time before that. Vendors had quietly been carving bits and pieces out of the reserved space for such things as network cards and a whole host of nonstandard devices. Because this was done surreptitiously, there were no rules and things just popped up here, there, or anywhere that their creators thought they could get by with a little benign poachery. As long as there was little competition for what seemed a lot of empty space up there (up to as much as 256K or more, depending on the display type), few users cared—or even noticed.

Despite IBM and Microsoft's best efforts to intimidate, coax, cajole, or otherwise head off the poachers, many otherwise reputable manufacturers started nibbling away at it for years, almost from day one—and ever more boldly as time went by when the Big Blue sky didn't fall.

Behold, the sky has fallen. It doesn't matter now who was poaching and who has legitimate rights above 640K. What matters now is that users somehow have to try to coexist with network cards, data compression cards, IPS cards, and all manner of things that have assumed squatters rights up there in the meantime. Even when it comes to legitimate usage, the boundaries get pretty fuzzy.

Take the 64K page frame typically starting at E000h as authorized by both the present and previous standards, for instance. IBM had claimed that space for ROM beginning with the AT, but the company never actually used it until much later, still confining themselves to F000h to FFFFh until the PS/2.

Obediently, Intel's AboveBoards stayed clear of the E000h page frame, but many expanded memory boards did and still do allow users that option—as do most memory managers. That is the top end of what EMS can use. The EMS page frame base address can be as low as C000h when necessary (if you don't have an EGA ROM, a hard disk ROM, or something else installed down there). The E000h address still is valid if your machine doesn't have somebody's ROM or something else sitting on those addresses up there.

There are really two ways to find out what's going on in your computer:

- Install something new, crash, remove the something new (because you can't reboot with it in there), then map address usage above 640, find the address conflict, see what can be moved to a different address (hopefully something can), and try again.
- Map existing address usage above 640K and look for possible conflicts before you start.

Unless you're a masochist, I suggest the second approach, which still doesn't address the problem of address conflicts below 640K—way down near the bottom where various devices access your system. To find out what's going on down there, you can choose between the same two approaches. In other words, you can do it now or you can do it later.

This is not to make light of a serious subject. However, you might as well laugh now, because it isn't very funny when it happens. This is especially true when dealing with 80386s, *i*486s and 286s equipped with memory managers like the ALL CHARGECARD, which allow you to map memory to any address below 1024K. This isn't to say you should be paranoid about it either, though. There are things you can do to make life easier.

There are clearly two different sets of problems: problems at high addresses and others at low DOS addresses. Most often, it is the high DOS area that users have to be concerned with. On newer machines, most low DOS problems are spotted when you first boot up your system and something doesn't match the CMOS configuration data.

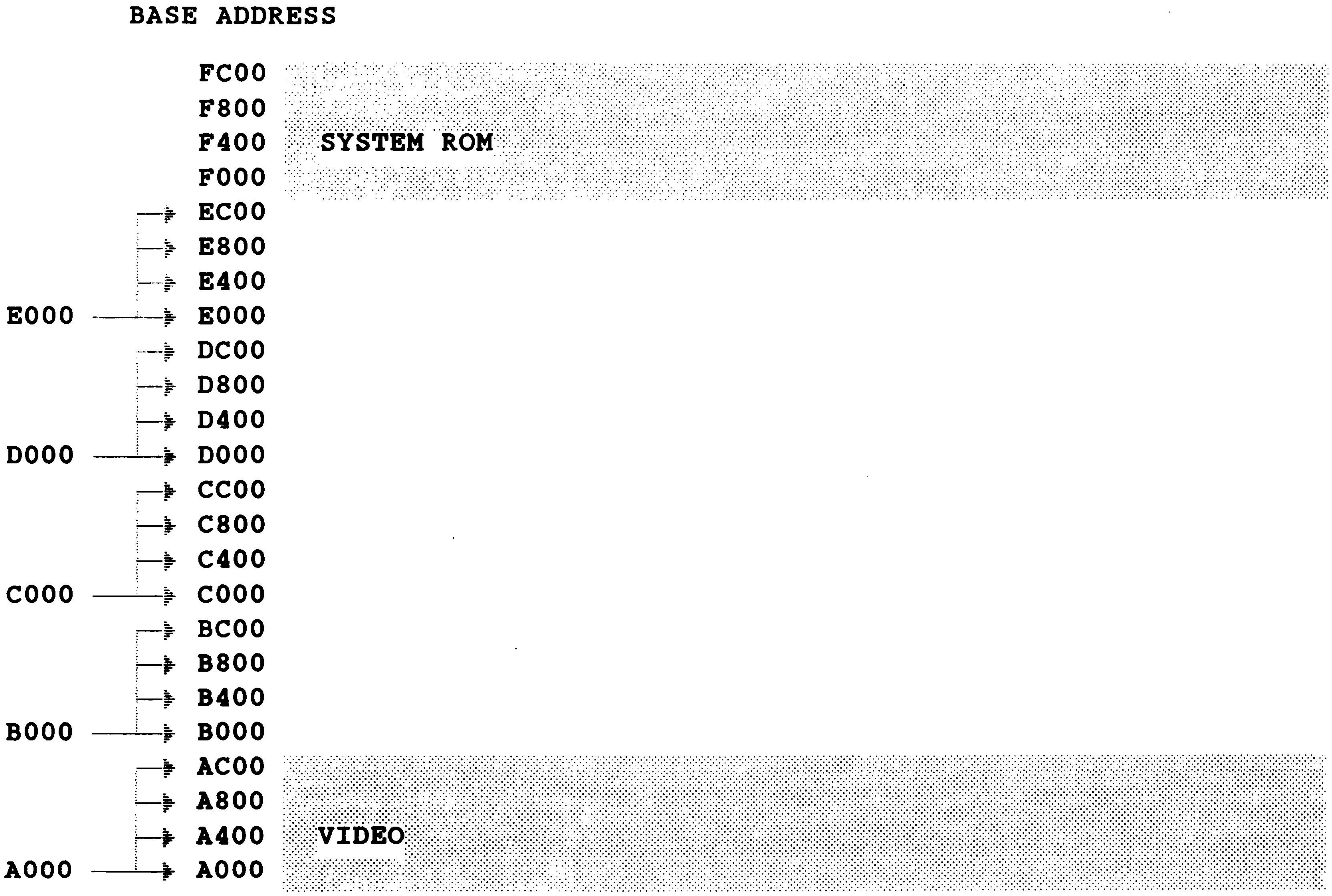
Upper memory, however, is another set of problems that demand a different set of answers. These are the problems—and solutions—of particular interest in the context of this book. I also will look briefly at the low end of the totem pole as well, however.

The high road

You would like to squeeze in everything you can up there. However, you'd better know exactly who and what is there and where before you start, because, for obvious reasons, no two things can occupy the same address. With remapping—or even without it—you can put them there. You won't be happy, though. You don't have to have an 80386 to find that out, even 8088's can play this game of musical chairs.

Suppose you've never used expanded memory before and now's the time. You bring the board home, open up the box, open your computer, and go to it. The Expanded Memory Specification allows page frames to start as low as C000h or at any multiple of 16K above that as long as the top is below F000h. That gives you a region that covers 192K to work in.

As shown in Fig. 18-1, C000h is above the 128K area set aside for video between A000h and BFFFh (EGA's only use the lower half of that). C000h looks like a good bet, so you try it and your system crashes.


BASE ADDRESS
FC00
F800
F400
SYSTEM ROM
F000
EC00
E800
E400
E000
E000
DC00
D800
D400
D000
D000
CC00
C800
C400
C000
C000
BC00
B800
B400
B000
B000
AC00
A800
A400
VIDEO
A000
A000

18-1 As far as you know, there's nothing in the way above 640K, as long as you stay clear of the video and ROM regions. Here, even if you were to allow 128K for the video—more than it needs—C000h would seem a safe place for the page frame, but don't bet on it.

If you had checked to see what else was up there before you started, you would have known that it wasn't going to work. There are a number of tools available to help you see what's happening up there, though many are device specific or work only in conjunction with certain other specific and generally proprietary software. A report can be obtained using the 386MAX options that only checks for ROMs, as shown in Fig. 18-2.

ROM In High Memory

Starting Address	Range Start	End	Length	ROM Option
000C0000	768	784	16	C000-C400
000C8000	800	808	8	C800-CA00
000F0000	960	1024	64	F000-10000

18-2 A 386MAX search of the address area above 640K revealed ROM in three different areas. It does not identify which devices have ROM located at those addresses, but that can be easily determined. These, however, are three areas that must be avoided when mapping memory to take advantage of unused address space.

Looking in high places—even on an 8088

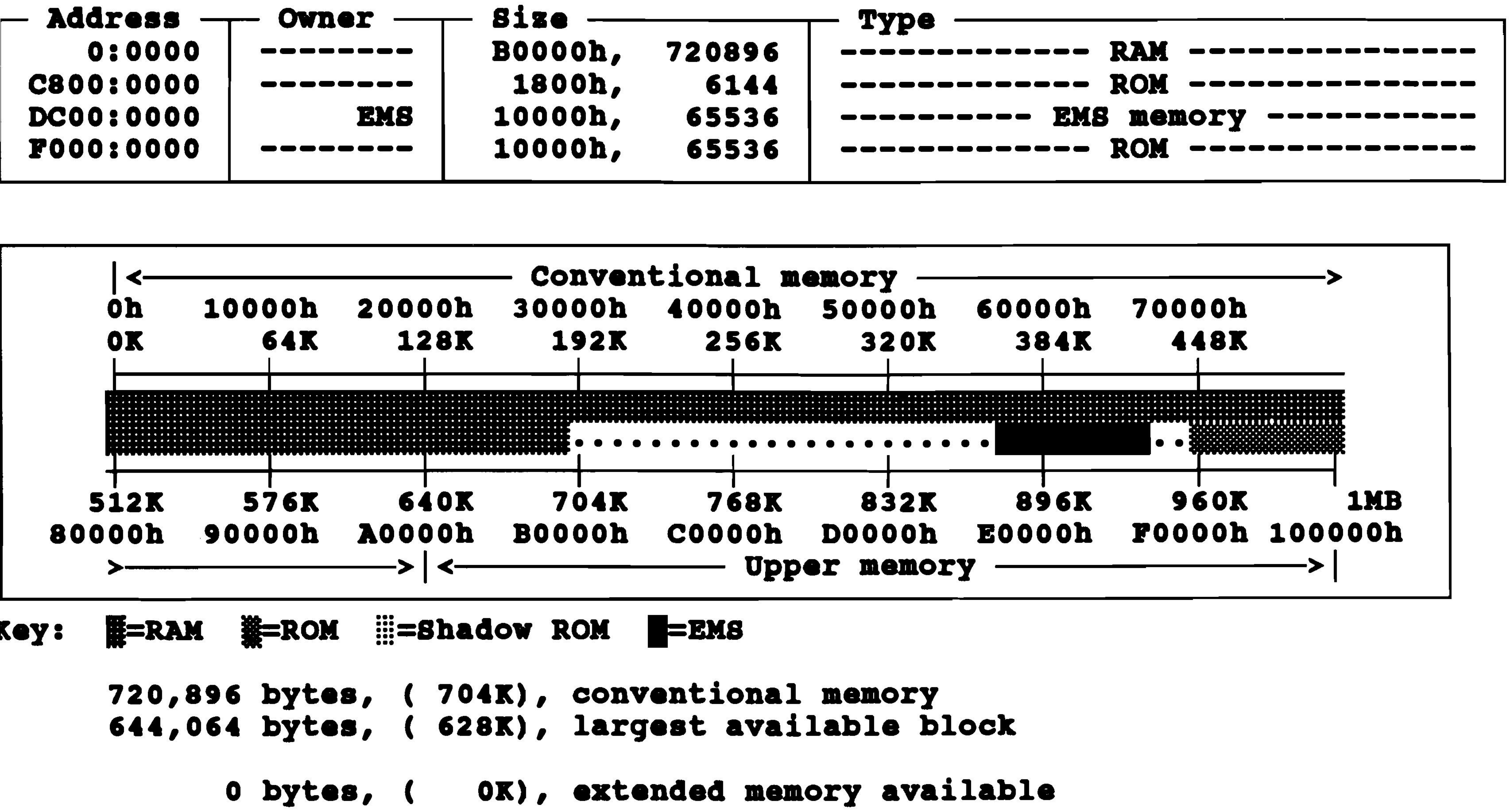
To be able to see and know what's going on inside your system, the hands-down winner is Quarterdeck's Manifest. It is sold separately and well worth the money at about $49. It also is generally bundled free with QEMM, QRAM, or DESQview 386. Although bundling with those memory managers might seem to imply that its usefulness was limited to 80286 and higher systems. It can be just as useful even working with an 8088—which is not the case with most of the others.

The MEM command in Digital's DR DOS 5.0 also can give you quite a bit of information even on an 8088 (Fig. 18-3). Interestingly, you can use this one when running under MS-DOS 5.0, but MS-DOS will not return the favor. Here, the start addresses and sizes are spelled out for both the hard disk ROM and the EMS page frame, which, in this case, is not exactly where you might expect it.

The MS-DOS 5.0 MEM utility, while critical to fine-tuning MS-DOS's HIMEM.SYS with EMS emulation (EMM386.EXE) on 386 and higher systems, is of little value to the 8088 user for spotting possible conflicts beyond 640K.

There are several third-party system snoopers on the market these days; however, of the ones I've tested, only one stands out as really being that much help, particularly working at the 8088 level. InfoSpotter from Merrill and Bryan is the exception.

This one is a little pricey, however. Unless you've got a bunch of PCs, you might better put the money toward a hardware upgrade. With more sophisticated



```
─ Address ─┬─ Owner ──┬─ Size ──────────────┬─ Type ─────────────────────────────
    0:0000 │ -------- │ B0000h,     720896  │ ------------- RAM --------------
 C800:0000 │ -------- │  1800h,       6144  │ ------------- ROM --------------
 DC00:0000 │      EMS │ 10000h,      65536  │ ---------- EMS memory -----------
 F000:0000 │ -------- │ 10000h,      65536  │ ------------- ROM --------------

  |<─────────────────── Conventional memory ───────────────────>
  0h    10000h   20000h   30000h   40000h   50000h   60000h   70000h
  0K      64K     128K     192K     256K     320K     384K     448K
  ├────────┼────────┼────────┼────────┼────────┼────────┼────────┼─────────
  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓..........................█████████..▒▒▒▒▒▒▒▒
  ┼────────┼────────┼────────┼────────┼────────┼────────┼────────┼────────┤
512K     576K     640K     704K     768K     832K     896K     960K      1MB
80000h   90000h   A0000h   B0000h   C0000h   D0000h   E0000h   F0000h 100000h
  >───────────────>|<──────────────── Upper memory ────────────────────>|

ey:  ▓=RAM  ▒=ROM  ░=Shadow ROM  █=EMS

     720,896 bytes, ( 704K), conventional memory
     644,064 bytes, ( 628K), largest available block

           0 bytes, (   0K), extended memory available
```

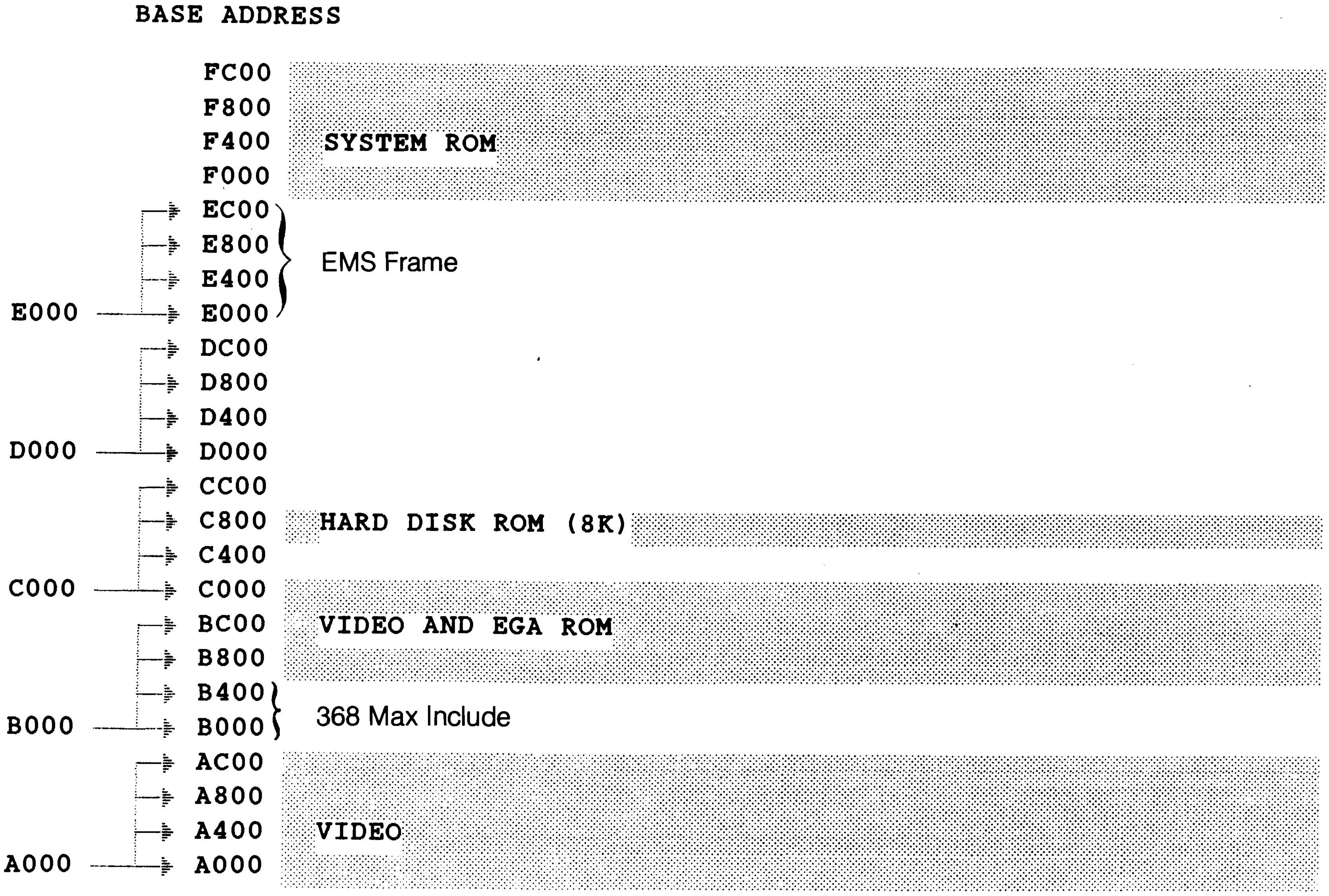
-3 The DR DOS MEM command displays address information clearly and shows usage in conventional and upper memory areas, depending on the switches used. The display here shows data from an 8088 machine running under DR DOS 5.0.

systems, this one has a lot to recommend it, so I don't mean to talk it down. Others, notably some with catchy names like Sleuth and CheckIt, really don't help with this kind of problem on an 8088 machine, although they do have other virtues.

In any event, more than just being able to determine the existence of devices that could cause possible conflicts when changing your configuration, you should keep an accurate record of every address block in use above 640K, whether that use is system ROM, a hard disk ROM, a network card, data compression card, IPS, or what have you. A simple chart, such as the one depicted in Fig. 18-4, should not only serve your immediate needs but also establish a framework for future needs as well.

You should never attempt to install a new board or change your hardware configuration without checking the documentation for any address needs the new device might have and checking those against current address allocations and resolving any possible conflicts before you crash your system. Believe me, a little time spent now can save you hours later on.

The need for such a record becomes increasingly important as you move on to 286 and especially to higher systems where the scene beyond 640K becomes increasingly crowded.


BASE ADDRESS
FC00
F800
F400
SYSTEM ROM
F000
EC00
E800
E400
EMS Frame
E000
E000
DC00
D800
D400
D000
D000
CC00
C800
HARD DISK ROM (8K)
C400
C000
C000
BC00
VIDEO AND EGA ROM
B800
B400
B000
368 Max Include
B000
AC00
A800
A400
VIDEO
A000
A000

18-4 This simple form shows addresses above 640K that are known to be in use by various devices. Something like this—either in a hardcopy stored with system documentation or stored for easy access in your computer—can be invaluable in preventing crashes and reducing downtime when you are installing new hardware (or software on 386 systems). Shading can be used effectively to mark out definite off-limits areas, but should be used sparingly to allow for changes.

On a higher plane

As you move on to 80286 and especially 80386 and higher systems, the need to know—ideally to be able to visualize what's going on above 640K—gets even greater. Most of today's more sophisticated memory managers—especially 386/486 memory managers but, within the limitations of the hardware, some of the better 286 managers as well—are smart enough to snoop around above 640K and sniff out at least a good part of the address space that can be mapped with memory.

However, as users have become increasingly conscious of the availability of upper memory and more TSRs and drivers have been written or updated to utilize upper memory when it is available, it has become increasingly crowded up there. To cope with the increasing need, better memory managers have become more aggressive in the methods used to find more memory to map.

This is really what you're paying for today when you buy QEMM, QRAM, ALL Charge 386, 386MAX, BlueMAX, or what have you, instead of simply using the management tools furnished with 5.0 (or higher) MS-DOS or DR DOS. As pointed out elsewhere, those management tools by and large have the raw power to do virtually anything you can do with the best of the third-party managers (with the notable exception of the symbiotic relationship between QEMM or QRAM and DESQview) if you know where to look for mappable address space.

Most users don't know. There really is little reason they should have to, not when some of the more powerful memory managers will even rewrite your CONFIG.SYS files, putting in INCLUDE= or USE= (or equivalent statements) to be sure you don't lose out on the benefits of some of their more aggressive rummaging.

However, this does not mean that you can simply assume that, having bought the best, you can just let them rip, because the problems first seen at the 8088 level are still there—some of them have even compounded. All that really has been done is that another layer of problems and potential problems has been added.

In addition to much more sophisticated memory managers to work with, you also have more tools at your disposal to help you visualize your situation—although most are keyed to a specific memory management package (even Manifest to a great extent). Still, even as good as the best of them are, you cannot get complacent.

Where did that come from?

As nice as it can be to be able to see what's going on upstairs with Manifest or one of the other comparable utilities, it is surely equally important that you understand their limitations. Good as they might be, none of them are perfect. Like the old deodorant ad said, there are things even your best friend won't tell you. Here, the issue is the fact that memory managers and their attendant viewers, if left on their own, can call the shots only the way they see them. A lot of things are not so easily seen—especially if they're not there to see.

Here, in a typical example, I have data from a Manifest report that was taken just after the system has booted and about halfway through the execution of the CONFIG.SYS. You can't actually run Manifest halfway through the CONFIG.SYS, so the last part of the CONFIG.SYS was taken out for the purpose of this demonstration. However, it does reflect the situation exactly as the system sees it at that point.

Given a free hand, QEMM—its companion memory manager and one of the more aggressive ones—has looked around above 640K and found all kinds of space that it can map memory to (it does this automatically every time it loads). In this case, as shown in Fig. 18-5, that space includes a 16K block beginning at C800h. Then, for some reason, it has skipped over a 4K block at CC00h—marked

```
A000 - AFFF    64K   VGA Graphics
B000 - B7FF    32K   High RAM
B800 - BFFF    32K   VGA Text
C000 - C7FF    32K   Video ROM
C800 - CBFF    16K   High RAM
CC00 - CCFF     4K   Unused
CD00 - DFFF    76K   High RAM
E000 - EFFF    64K   Page Frame
F000 - F7FF    32K   System ROM
F800 - FDFF    24K   High RAM
FE00 - FFFF     8K   System ROM
     HMA       64K   First 64K Extended
```

18-5 At bootup, the memory manager—any memory manager—is unable to anticipate the address space requirements of the device drivers that will load later in the boot cycle. In this case, it has mapped high RAM to 16K to an area reacquired by a device that must load at C800h. Such conflicts must be anticipated by the user on the basis of device documentation and must be resolved by placing specific exclusion statements on the memory driver command line.

as unused by Manifest—but then mapped the next 76K. For some reason, it has broken what appears to be a contiguous 96K block. That big of a block would be a real beauty up there.

Apparently, the memory manager found something it didn't like at CC00h. Indeed, there is a data compression board sitting there, but that board actually requires not 4K but 20K. The problem is that there was nothing to tell QEMM—or Manifest—that it has to have 20K to operate, certainly not at this point in the boot cycle, because at this point, the need does not exist. The device driver wasn't even loaded yet. Even if it was, there likely wouldn't be a flag that said "keep off." So, QEMM—like any reasonably aggressive manager worth its salt—has simply assumed that it could have all but that 4K block.

Here, you'll note that even Manifest can't tell what is there and simply marks that block as unused. What QEMM found there was just the signature of the compression card, as shown in Fig. 18-6, which was all it needed to know that it had to stay clear of that block.

```
debug
-dCC00:00
CC00:0000  41 53 48 45 52 00 00 00-00 00 00 00 00 00 00 00   ASHER...........
CC00:0010  46 69 72 6D 77 61 72 65-20 52 65 76 69 73 69 6F   Firmware Revisio
CC00:0020  6E 3A 20 20 31 31 20 20-31 31 2F 32 32 2F 38 39   n:  11  11/22/89
CC00:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

18-6 Checking the beginning of the address block that QEMM didn't map with MS-DOS DEBUG, you find the signature of a piece of hardware.

Just to prove there was something in the way there, I checked with DEBUG (DR DOS's SID would have done as well). There is, indeed, something there. A check of the documentation for the intruding device shows that 4K is only just the tip of the iceberg.

This is the point at which you have to intercede and tell whatever memory manager you might be using that it has to stay clear of a whole 20K block that has a much lower starting address, based on information in the documentation for that hardware and its device driver, as shown below:

```
DEVICE=C:\DOS\QEMM\QEMM386.SYS RAM ROM
    EXCLUDE=C800-CCFF AU DMA=32
```

While the exact syntax varies, all worthwhile 386 memory managers (or others capable of mapping to high DOS address space) provide both a means of blocking off (excluding) certain areas to prevent conflicts and a means of adding (including) blocks that the defaults would ignore. To achieve the same degree of mapping with MS-DOS 5.0's HIMEM.SYS/EMM386.EXE duo requires not only the same exclusion (on that machine with that configuration) but also two inclusions (I=*xxxx-yyyy*), as shown here:

```
DEVICE=C:\DOS\HIMEM.SYS
DEVICE=C:\DOS\EMM386.EXE 256 RAM I=B000-B7FF
    I=F800-FDFF X=C800-CBFF
DOS=UMB
```

Let me again stress that this behavior is characteristic of all memory managers capable of mapping memory to address space above 640K. I used QEMM to demonstrate earlier primarily because, aside from some managers that now provide BIOS compression on PS/2s, it is one of the more aggressive managers and its companion viewer, Manifest, shows the situation so clearly.

Admittedly, there is a lot to keep track of; however, it really isn't all that hard if you approach it systematically. With tools like Manifest, the 386MAX utilities, and ALLMenu, you don't even really have to chart things manually. You don't even have to when using either of the DOS 5.0s or later memory management systems. You do need to keep a record of your current and possibly recent past configurations. To make life easy, Manifest, AllMenu, InfoSpotter, and some of the viewers and tabulators can send the data directly to a printer or a file for you.

Some of the others—like the MS-DOS or DR DOS MEM commands—don't have that capability built in. However, the output of any character-based utility can be redirected either to a file to print right from the command line, as shown below:

```
MEM /m >e:memfile   or
MEM /m >PRN         (or LPT1 or COM1 as required)
```

This is the exact command and syntax used with the DR DOS MEM command to redirect the DR DOS data shown earlier. It is a trick I use a lot for capturing screens.

Before I leave this subject, I'll give you just one more sage bit of advice: never make more than one configuration change at a time—hardware or software.

I know it's tempting sometimes—just some little thing you've been meaning to do—but don't.

Any old port in a storm

Any port in a storm might be good advice for sailors, but it only can get you in hot water if you take that approach when selecting an address port for devices that give you the option of picking one. Ports are another one of those things that most users talk more than they know about. However, if I might digress briefly from the heady world above 640K to look at more mundane matters, address conflicts are by no means limited to upper memory.

Users take such things as parallel and serial ports for granted; however, like everything else a computer has or does, such things must have base addresses, too. However, unlike the typical 4K minimum block size that users deal with in high RAM, ports occupy much smaller chunks of memory—often just a few bytes wide. They often are not as neatly—or uniformly—defined either. The blocks are small and largely arbitrary.

It isn't as bad as it might seem, though. Maybe it's just me, but these conflicts don't seem to occur as often as they used to. Also, some of the smarter device drivers are able to hunt through the port address space until they find a vacant place to install themselves.

In any event, conflicts still do happen occasionally. You should know how to deal with them. The key to preventing such conflicts—or to resolving them, if they do occur—is exactly the same as in the high DOS area:

- Keep track of the address of any nonstandard devices already installed. This list also should contain data on alternate addresses that can be used with those devices, if it is necessary to move them.
- Check the documentation for whatever new device you are installing for possible address options.
- Check your current DOS configuration for existing port addresses, then check against addresses being used by other nonstandard devices.

Port conflicts generally are less likely to crash your entire system—which is just as well, because port assignments are usually made before the computer is ready to turn control over to the operator. Symptomatically, one or more devices simply will not come on-line, which might or might not result in an error message being generated and reported on your display.

Software crashes can be just as hard

Although hardly unique to using extended and expanded memory, the likelihood of system crashes seems to increase exponentially as users are able to load at the same time in all that extra memory. Tracking down the culprit when you start having more than your usual share of system crashes can be a frustrating business, doubly so if you do not approach it systematically. Your immediate conclusion most often is likely to be that it is whatever application you are running, but it isn't necessarily.

It could be a TSR kicking its heels because it didn't like something your application did. It could be something about the DOS version you're using in combination with your particular hardware and software. As you'll see shortly, it isn't always as easy as you might think, but it isn't necessarily that difficult either, if approached properly.

This is not to say that every time your system has the hiccups you should panic and start looking for a major problem. In all those millions of bytes, some random things can happen sometimes. It's only when it seems to happen often or when there seems to be a pattern that you start to look for trouble.

Often your system can be extremely helpful if you let it. Take for instance something like:

```
Exception Error #13 at 1607:0000
Error code 0000
Do you want to T)ermiante the program, R)eboot, or try to C)ontinue?
```

When you get a message like this, make a note of it before you reboot or do anything—not verbatim necessarily, but at least the numbers. The important part in most instances other than the random case is the address. Even if you knew what Exception 13 or Error Code 0000 were, it probably wouldn't help much—certainly not until you knew which of the various programs you had loaded was having the problem. That information, however, could be very helpful when you call to report the problem to someone's technical support staff.

You don't know which program caused the problem, but at least in this case, you've got one number you can go to work with. You have an address: 1607h. That address means the problem is down in low memory.

This actual case history example turned out to be a rather easy one, but the method would be the same any time your system was kind enough to leave a clue like that as it died. What's needed at that point is a program you can run (before you crash) that reads and reports what is loaded where. This also requires that your applications allow you to access DOS while still in them so that your map reflects the total situation including the application that you're in. Not all applications will shell to allow you to access DOS to map your system from within them, but fortunately many will.

Some of the more sophisticated memory management programs include mapping utilities. There also are a variety of freestanding mapping utilities that can be run under Microsoft Windows or DESQview or in conjunction with other windowing environments and/or applications software. Figure 18-7 shows a typical

```
PCMAP 1.0 (c) 1987, Ziff-Davis Publishing Corp.
Segment Paragraphs    Bytes      Program
1517H      00D1H       3344      COMMAND.COM
1E58H      0375H      14160      DV.COM
1E34H      001FH        496      UNCRASH.COM
1E0FH      0023H        560      MODE.COM
19C7H      0446H      17504      VKETTE.COM
15F7H      03CEH      15584      CULPRIT.COM
1E78H      0003H         48      command.com
21D3H      00DAH       3488      command.com
22A2H      0004H         64      (Free)
22B4H      0A2BH      41648      SQLOAD.EXE
2CE1H      0065H       1616      CHSTACK.COM
2D48H      335EH     210400      EDITOR.EXE
60A7H      00D8H       3456      COMMAND.COM
6183H      3E87H     256112      PCMAP.COM (Free space)
```

18-7 An address map showing CULPRIT.

report from one such freestanding program. This report is from PCMAP.COM, one of the free programs available for downloading from PC magazine's bulletin board; however, it is compatible only through DOS 3.3

Note that there is no program neatly flagged AT 1607, Rather 1607 is an address somewhere within the block occupied by CULPRIT.COM (I took the liberty of renaming it after I had it identified). With a little simple arithmetic, you find:

$$1607h - 15F7h = 10h$$

That's just 16 bytes above the starting address of CULPRIT—a disk caching program in this case—as reported by MEM /d, Manifest, or any program you have handy that can give that information. Because it is loaded from the AUTOEXEC.BAT, you would expect it to always show up at the same address.

The proof comes when you remove CULPRIT from the loading sequence and see if the problem goes away. In this case, the culprit was an old favorite that took an immediate dislike to something new in the system. What you do about it is up to you. In this case, a call to the culprit's creator was answered with a quick upgrade release via return mail. Other people, it turned out, had the same problem.

Problems won't always be in low memory. Things can go sour up in high memory, too. This is especially true when you try to squeeze out that last possible block of unused memory up there. However, just as there are mapping programs for low memory, there are utilities that read and report what's where between A000h and FFFFh.

Unfortunately, there are some devices—typically graphics adapters—that use pieces of legitimate high RAM as what might be called scratch pad space. No amount of memory mapping will point to these addresses which, by their nature, leave no signature. If you try installing something else in seemingly unused address space, you might stumble onto one of these. If you do, you're going to have a crash. In the case of a graphics board using scratch pad space, you'll likely find yourself with no display.

A special word about tracking down problems that might involve programs running in expanded memory in windowing environments like DESQview and Windows (or context-switched as under Carousel): the addresses reported are DOS addresses. What programs actually are using those addresses changes every time you switch windows however—the same programs, always coming up at those same addresses within a work session (or loaded in the same sequence in subsequent sessions).

While in the case cited here, the offending program was below the base address of any of the DESQview windows that were open at the time, it might not have been. In that case, only a map that reflected the system including the software in the window that was open at the time of the crash could give any real clue to the cause. If you have to change windows to access DOS, the act of switching windows will hide the application somewhere in expanded memory where you can't map it.

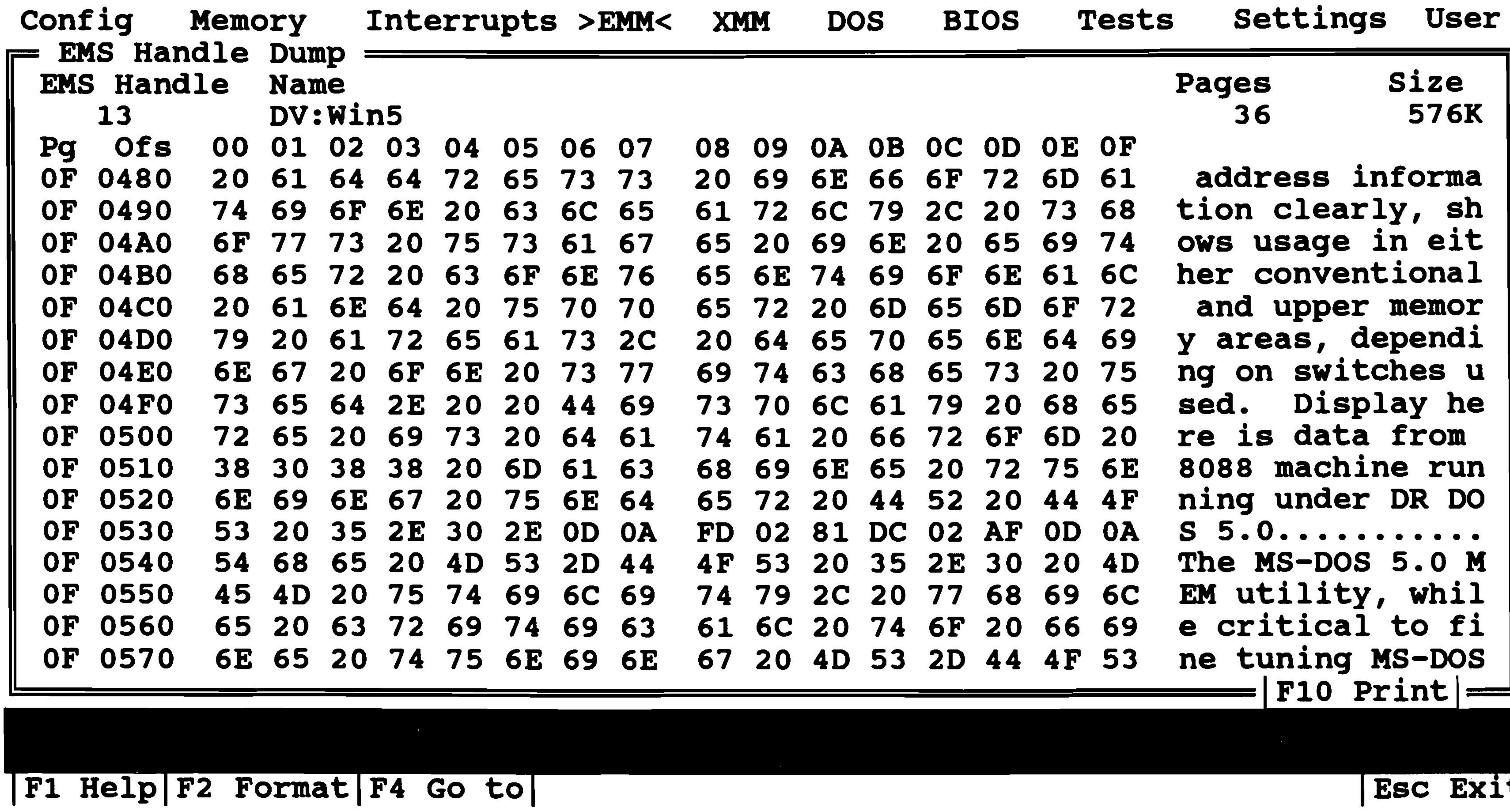
Detectives on a disk

Several third-party system snooping software packages have hit the shelves, promising—and generally delivering—all sorts of system information. Unfortunately, little of the information is germane to issues dealt with in this chapter—or even in this book. Even where the information is valid, it generally is obtainable with MS-DOS (or DR DOS) utilities like DEBUG (or SID) or MEM with one or more of the various supported switches.

Of the lot, about the only one I've found so far that warrants serious attention is called InfoSpotter. Aside from providing system data, under certain circumstances, it can recover lost data from the far hinterlands of memory. It doesn't always work, but it doesn't have to very often to pay for itself. It's certainly worth a try.

To be recoverable through InfoSpotter, the handle for the host program must not have been released. You cannot have exited the program and suddenly discovered five minutes later that you didn't save what you were working on and try to go back to it. The data still is floating around out there somewhere; however, once it is returned to the memory pool, there's no practical way of ever finding it, let alone recovering it. However, if you have accidentally aborted a data file or, in some cases, hung the system with at least part of the data unsaved you still might have a shot at it—at least with text files.

My word processor, for example, allows up to nine windows to be open at any time. More than once, one has been invertently closed. Until now, whatever pearls of wisdom it might have contained were lost—forever usually, because it's awfully hard to exactly reconstruct the process that led to writing them. After aborting a window, however, I was able to recover the data shown in Fig. 18-8.



```
Config   Memory   Interrupts >EMM<  XMM   DOS   BIOS   Tests   Settings  User
 EMS Handle Dump
 EMS Handle  Name                                                Pages     Size
    13       DV:Win5                                               36      576K
 Pg  Ofs   00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F
 0F 0480   20 61 64 64 72 65 73 73  20 69 6E 66 6F 72 6D 61   address informa
 0F 0490   74 69 6F 6E 20 63 6C 65  61 72 6C 79 2C 20 73 68  tion clearly, sh
 0F 04A0   6F 77 73 20 75 73 61 67  65 20 69 6E 20 65 69 74  ows usage in eit
 0F 04B0   68 65 72 20 63 6F 6E 76  65 6E 74 69 6F 6E 61 6C  her conventional
 0F 04C0   20 61 6E 64 20 75 70 70  65 72 20 6D 65 6D 6F 72   and upper memor
 0F 04D0   79 20 61 72 65 61 73 2C  20 64 65 70 65 6E 64 69  y areas, dependi
 0F 04E0   6E 67 20 6F 6E 20 73 77  69 74 63 68 65 73 20 75  ng on switches u
 0F 04F0   73 65 64 2E 20 20 44 69  73 70 6C 61 79 20 68 65  sed.  Display he
 0F 0500   72 65 20 69 73 20 64 61  74 61 20 66 72 6F 6D 20  re is data from
 0F 0510   38 30 38 38 20 6D 61 63  68 69 6E 65 20 72 75 6E  8088 machine run
 0F 0520   6E 69 6E 67 20 75 6E 64  65 72 20 44 52 20 44 4F  ning under DR DO
 0F 0530   53 20 35 2E 30 2E 0D 0A  FD 02 81 DC 02 AF 0D 0A  S 5.0...........
 0F 0540   54 68 65 20 4D 53 2D 44  4F 53 20 35 2E 30 20 4D  The MS-DOS 5.0 M
 0F 0550   45 4D 20 75 74 69 6C 69  74 79 2C 20 77 68 69 6C  EM utility, whil
 0F 0560   65 20 63 72 69 74 69 63  61 6C 20 74 6F 20 66 69  e critical to fi
 0F 0570   6E 65 20 74 75 6E 69 6E  67 20 4D 53 2D 44 4F 53  ne tuning MS-DOS
                                                                  |F10 Print|

|F1 Help|F2 Format|F4 Go to|                                         |Esc Exi
```

18-8 A fringe benefit of InfoSpotter is its ability to help you recover lost data from memory limbo, under certain circu stances.

Granted, this is only 256 bytes at a time or about 50 words of text. However, by starting at the beginning of the lost portion and appending succeeding screens to the file (InfoSpotter will do that for you), you can recover as much as you like. If it was in memory, it's there. With the help of a little macro to strip away all but the good stuff, it will take a lot less time than trying to redo it. Once you've opened a new file, all bets are off; however, there is a moment you just might have one last shot at it with InfoSpotter.

In fairness, I should quickly point out that you can do the same thing using MS-DOS's DEBUG (version 4.0 or later) or any comparable utility you have at hand. It's mainly just a matter of mapping logical pages for the handle in question to physical pages. Assuming you're using MS-DOS, at the DEBUG prompt, enter:

XM *ll pp hhhh*

where:

ll is the number of the logical page to start with
pp is the number of the physical page to map to
hhhh is the number of the handle

The handle number is obtainable with MEM /d, Manifest, or any of at least a dozen other utilities. Beyond that, it's just a matter of doing your DEBUG thing, with the added advantage of being able to use DEBUG's search command to help you zero in on just the data that you're looking for, as in this example:

```
snnnn:nn rrrr "string"
```

where:

nnn:nn is the starting address of the block you want searched
rrrr is the range (in hex)
string speaks for itself

This takes a little more experience than using something that's more user friendly. It also is much more flexible and powerful.

To track down the most common kinds of problems both above and below 640K, it's really hard to beat the utilities supplied by the major players in the memory management game or with DOS when it comes to analyzing system resources and putting them to work. When all else fails, they give the information necessary to resolve whatever problems they can't handle. When it comes to that, you can use all the help you can get.

19
CHAPTER

Parting shots

I have discussed many aspects of the world beyond 640K; however, I have barely scratched the surface. Much more is heating up than just the war between the competing DOSs—one in which Microsoft, if only by virtue of force of habit, draws the most attention, remains preeminent in the marketplace, and thereby moves the world—the DOS world anyway.

As soon as MS-DOS 5.0 was officially announced, the rumblings of another salvo could be heard as Digital prepared to answer MS-DOS 5.0 with DR DOS 6.0, offering new features will not seen in MS-DOS and going one step farther yet in memory management by breaking up the kernel into more than just the single 38K piece that could be loaded high in DR DOS 5.0, even into rather badly fragmented upper memory on 80386 and higher systems, and extending memory mapping support to other chip sets now in use on many 80286s.

The activity has not been limited to just those two contenders. Few things are truly secret in this industry for long, especially as beta copies of upcoming software packages begin to circulate (Microsoft put out MS-DOS 5.0 beta copies to something over 5000 sites starting almost a year before its ultimate release). As a result, not only was this probably the most thoroughly debugged release in history, but other developers had a lot of time to look for—and fix—any incompatibilities the new DOS might exhibit. A lot of people got a jump on writing or revising software with MS-DOS 5.0 in mind.

Quarterdeck, for instance, already had a major new 6.0 release of QEMM—updated to exploit MS-DOS 5.0's shortcomings—ready for release almost as soon as MS-DOS 5.0 started showing up on dealers's shelves. Suddenly, there was a rush of new beta offers and my FAX machine was busy with a spate of nondisclosure forms for those of interest.

The cycle starts anew; the software wars go on. Ultimately it is the users who are the winners as the two DOSs face off, each growing stronger with each round that's fired. Yet, if the pundits had been even partly right, everyone would have all forsaken DOS long ago in favor of some new and far more powerful operating system.

Ship of fools

Events have a funny way of making fools of everyone sometimes. One of my favorite stories, left over from my childhood, is about nine blind wise men who had never seen an elephant and went to study one. Each encountered a different part of the elephant's anatomy: one its snakelike trunk, another its spearlike tusks, another its leaflike ear, and yet another the north end of the south-bound beast—a tail resembling frayed rope. They went away, each arguing a quite different thesis on the nature of an elephant.

To a great extent everyone is like those blind wise men, for this field is far too vast for anyone to grasp the total picture. Users can analyze only bits and pieces in detail and then ponder and, unable to see the total picture, draw conclusions from that which might or might not be valid in the total context.

In the short span of years between the first edition of this book and this one, the insatiable appetite for memory has continued to increase about tenfold every five years and shows no sign of slowing. Fortunately, the cost of memory has fallen from about a buck per kilobyte at a commensurate rate, which has largely kept the new technology affordable.

To stretch a point

In discussing software in this book, I have largely glossed over one of the real stars of the show: the DOS Extender, the vehicle by which today's more powerful software can escape to take full advantage of the 16-bit and 32-bit CPUs that are so much for granted these days. This is not an oversight, however, because, in the context of what you need to know to live and work not only just beyond 640K but far beyond, there is really very little users need to know about the DOS Extender.

It is nice to have a general understanding of the role of DOS Extenders. It is nice to know that DOS-extended programs generally run faster—often significantly faster—mostly by virtue of being able to unleash the full processing potential of your machine, but they also are unencumbered by DOS's segmented memory model.

However, they are—and always will remain—invisible to you. They require no special consideration when you configure your machine. They only require that you provide the necessary hardware platform—32 bits for many DOS extended programs and at least a 286 or better CPU for the rest, all with memory resources sufficient for their needs.

In that regard, you should know that many—most DOS-extended programs probably—can virtualize memory (swap to disk) to run in less real memory than they would like to have available. Swapping to disk generally will result in some degradation in performance, but it might not even be noticeable, depending on how well programs lend themselves to being broken into modules that, a little on the order of overlays, can be swapped out and how much swapping is required because of memory constraints.

True memory virtualization represents a sophisticated technology quite different from the arbitrary brute-force and dead-in-the-water kind of swaps-to-disk behavior associated with crude implementations, like MS-DOS 5's new task swapper. At some point you will see a difference probably, a noticeable slowing; however, as long as you have enough memory to meet the bare minimum requirements of your software, at least you're up and running. You can always add more memory if you are inconvenienced. Beyond that, from a user point of view, a DOS-extended program really is remarkably unremarkable except to the programmer.

The more things change

The more things change, the more they really stay the same in many ways. There have been changes and there will continue to be changes as the technology improves; however, in terms of memory management, what has been seen is not so much of a change but a general realignment in which one or more of the new DOSs has embraced at least the substance, if not the essence, of memory management techniques pioneered by third-party developers, most notably by Quarterdeck but with others playing major roles as well.

When I say the "substance if not the essence," as you have seen here, it is not the raw power to manage memory that is lacking in either MS-DOS 5.0 or DR DOS 5.0. What is lacking most in both is the sophistication necessary to explore your system's resources and go beyond a rudimentary set of safe and simple, but too often inadequate, defaults to exploit them adequately.

For this, you must still look to the third-party market. It seems unlikely this will change in the foreseeable future. It is not in Microsoft's or Digital's best interest to kill that market or suppress the kind of creativity that the market spawns—from which they too will ultimately benefit.

Multiuser DOS and DOS-like systems dramatically change the equation in the way they utilize whatever usable address space you have available beyond 640K and make their own rules tailored to their own specific needs. I have briefly only touched on how that might impact upon your DOS habits in the workplace and completely overlooked the subject of full scale Local Area Networking. Both areas are sufficiently complex to warrant dedicated volumes of their own.

The ultimate shell game

In what could be another of those strange twists of irony, regardless of the ultimate fate of windows, its true legacy might be the DPMI that Bill Gates pushed so hard for as a means of keeping Windows in the game with the coming of the DOS Extender.

The real promise of the DPMI, if it indeed sees fruition, is that it has the potential to cut across operating system boundaries and to allow conforming DOS-extended applications programs to run not only under DOS but under OS/2, UNIX, and possibly other essentially incompatible operating systems as well.

Yet at the same time, while everyone's attention has been riveted on DOS and all that's new—and yet not really new—beyond 640K, another pot is coming to a boil. It looks like OS/2 really is coming this time. Despite the problems that have plagued it from its earliest beginnings, OS/2 still keeps resurfacing. Each time it does, it is enhanced a little more and is less hostile to the DOS that it is trying to replace.

After the much talked about split between Microsoft and IBM (apparently the result of a furor over Windows), OS/2 2.0 is coming now from IBM. This time, it is not just a curiosity, an orphan lacking interest or support from any quarter, but a robust new 32-bit operating system with the power to embrace today's DOS applications. Claimed to be a better DOS than DOS and a better Windows than Windows, it will even multitask the applications now run under DOS. No longer limited to the single session compatibility box that never really was compatible, OS/2 intermixed with super-powerful new applications written to exploit processing power that DOS can never equal.

In any event, one thing seems very clear as DOS enters its second decade. The future of the familiar DOS environment has been assured now for some time to come. The most intriguing—and exciting—irony is that, as the role of operating systems changes and the boundaries become increasingly unclear, it might not even be a DOs that takes users there.

Regardless of how everyone gets there, it is memory technology itself that is the real growth industry, not DOS or OS/2 or UNIX. Those are simply vehicles used as means of accessing and manipulating computer memory. It is the memory that is important. It is at least equal in importance to the CPU, because without memory, a lot of memory—more and more each day it seems—the CPU is meaningless. In this age of gigabytes and even terabytes, the steps taken beyond 640K today are only the beginning.

A

APPENDIX

Advanced programming functions available under LIM 4.0 EMM

The following list contains the functions available under versions 3.2 and 4.0 of the LIM specification for EMM. Functions 1 through 15 apply to LIM 3.2 only. The remaining functions (16 through 30) were added by LIM 4.0 EMS.

Number	Description
1	**Get Status** test whether the expanded memory hardware is functional
2	**Get Page Frame Address** obtains the segment address of the page frame used by the expanded memory manager
3	**Get Unallocated Page Count** obtains the total number of logical pages of expanded memory present in the system and the number of pages not already allocated
4	**Allocate Pages** notifies that the EMM program will be using expanded memory, obtains handle, and allocates the required number of logical pages to be controlled by that handle

5	**Map/Unmap Handle Page** maps one of the logical pages assigned to a handle into one of four physical pages within the expanded memory manager's page frame
6	**Deallocate Pages** releases the logical pages of expanded memory currently assigned to the handle, then releases the handle
7	**Get Version** returns the version number of expanded memory manager software
8	**Save Page Map** saves the contents of the page mapping registers from all expanded memory boards into an internal save area
9	**Restore Page Map** restores (from an internal save area) page mapping register contents on expanded memory boards for a particular EMM handle
10	(no longer used)
11	(no longer used)
12	**Get Handle Count** returns the number of open EMM handles in the system
13	**Get Handle Pages** returns the number of pages allocated to a specific EMM handle
14	**Get All Handle Pages** returns an array of active EMM handles and the number of pages allocated to each
15	**Get/Set Page Map** (subfunction) saves, restores the mapping context for all mappable memory regions (both conventional and expanded) in destination array supplied by application
16	**Get/Set Partial Page Map** (subfunction) provides mechanism for saving partial mapping context for specific mappable memory regions
17	**Map/Unmap Multiple Handle Pages** in single invocation, can map (or unmap) logical pages into as many physical pages as is supported by the system
18	**Reallocate Pages** can increase or decrease the amount of expanded memory allocated to a handle
19	**Get/Set Handle Attribute** allows the application program to determine and set the attribute associated with the handle

20 **Get/Set Handle Name** gets eight character name currently assigned to the handle and assigns an eight character name to the handle

21 **Get Handle Directory** returns information about active handles and names assigned each

22 **Alter Page Map and Jump** alters memory mapping context and transfers control to the specified address

23 **Alter Page Map and Call** alters specified mapping context and transfers control to the specified address. A return can then restore the context and return control to the caller

24 **Move/Exchange Memory Region** copies or exchanges a region of memory from conventional to conventional, conventional to expanded, expanded to conventional, or expanded to expanded memory

25 **Get Mappable Physical Address Array** returns an array containing the segment address and physical page number for each mappable physical page in the system

26 **Get Expanded Memory Hardware Information** returns the array containing hardware capabilities of installed expanded memory

27 **Allocate Standard/Raw Pages** allocates the number of standard or nonstandard size pages that the operating system requests and assigns a unique EMM handle to these pages

28 **Alternate Map Register Set** enables an application to simulate alternate sets of hardware mapping registers

29 **Prepare Expanded Memory Hardware for Warm Boot** prepares expanded memory hardware for "impending" warm boot

30 **Enable/Disable OS/E** enables operating systems developers to enable and disable functions designed for operating system use

B

APPENDIX

Special considerations for mapping LIM 4.0 EMS memory

In accordance with the LIM 4.0 specification, all of the better memory managers for 80386 and higher systems provide for mapping memory. There are, however, a number of programs that have problems if they encounter mapped memory in specific areas that they are programmed to try to use, including a number that (at least with certain memory managers) have difficulty dealing with mapped memory below 640K. In these cases, special allowances must be made when setting up the memory manager. These allowances usually entail including some sort of specific EXCLUDE statement on the memory manager's command line in the CONFIG.SYS, as in the example shown here:

```
DEVICE=QEMM386.SYS ram rom EXCLUDE=C800-CBFF au dma=32
```

Although, in many cases, other command line parameters are determined and written to the CONFIG.SYS automatically by the installation program, such exclusions generally have to be added by the user. Information relative to the need for special address exclusion should be part of the documentation for any program requiring such special treatment. However, if any conflicts are observed from error messages during loading or the malfunction of any software that had been working previously, it might be necessary to call for technical support.

Difficulties encountered by the failure to exclude specific address areas vary. In some cases, it results only in programs that might otherwise use high memory being loaded in conventional memory instead. In more severe cases, it can result in the failure of certain hardware or software to function.

The following listing presents a sampling of programs known to have encountered problems under certain circumstances and exclusions or other corrective measures that can be taken to ensure proper operation. I have used EXCLUDE= and FRAME= in the examples shown below. Although this is the syntax used by several memory management packages, you should refer to the documentation for the specific manager you are using to determine the syntax that should be used. If a problem does exist, the specific address areas should be constant regardless.

Program name	**Special instructions**
DC Windows Express	EXCLUDE=1000-A000 (under Windows 2.*x* or use WIN/N)
Deluxe Paint	EXCLUDE=1000-A000 (for versions 1 and 2)
Excel	EXCLUDE=1000-A000 (run-time version only)
Javelin	EXCLUDE=1000-A000
Pagemaker	EXCLUDE=1000-A000 (under Windows run-time 2.*x* or use WIN/N)
Paradox 3.0	EXCLUDE=A000-B000 (for monochrome or CGA systems)
Smartware II	EXCLUDE=1000-A000
SQL Windows	EXCLUDE=1000-A000
Stacker	EXCLUDE=C800-CBFF
SuperCalc5	EXCLUDE=1000-A000 (see the text at end of this chart)
SuperProject+	EXCLUDE=1000-A000
Word for Windows	EXCLUDE=1000-A000 (under Windows 2.11 and 2.1)
JLASER/SA	FRAME=D000 (versions of the JLASER/SA software prior to 4.14 only)
Ventura Publisher	FRAME=E000 (or below if the EMS page frame starts above E000)

There are also some programs that, although normally run in color mode, can use the monochrome buffer area to access special characteristics of the VGA/EGA card installed. If these programs are used, the monochrome buffer area cannot be reclaimed as high DOS or as an EMS without video problems resulting.

Of these programs, SuperCalc5 and Smartcom are probably the best known, although there are others. In such cases, be aware of any statements on the memory manager's DEVICE= line that would include the B000 to B800 area. If any are found, those references should be deleted.

Additionally, a number of display adapters are known to require special parameters to be set when the adapters are used in conjunction with certain memory managers. When installing new display adapters (particularly special purpose or high resolution adapters), special attention should be given to the documentation in this regard.

C
APPENDIX

Using bootable floppies to test new configurations

When experimenting with upgrades to new DOS versions, alternative operating systems, or different memory managers, it is virtually imperative that you leave whatever system you are using intact on your hard disk. Do as much experimentation as possible working from bootable floppy disks rather than blindly installing something that may take hours, even days, to fine-tune to your system—if you even can fine-tune it to meet your particular needs.

Getting up bootable floppies is now pretty much down to a science. At any given time, users typically have a dozen or more of them, each with some different combination of DOS versions and/or alternate operating systems working in combination with different memory managers. Some bootable floppies are fine-tuned nicely; however, others are not. User needs are different, but the problem is always the same. Even when you think you've got things worked out nicely, you get bit once in a while when you finally try installing something to your hard disk. No system is infallible.

There are still some basic bootable floppy strategies I'd like to pass along. First, you probably cannot simply copy your existing CONFIG.SYS and AUTOEXEC.BAT files to a bootable floppy and automatically have them work. Chances are you've got lines in your AUTOEXEC.BAT like:

```
MODE COM1:9600,N,8,1,P
```

If you boot from drive A; with MODE.COM residing in C:\DOS, for example, you're going to have a problem. One of the easiest ways around this problem is simply to add a line at the top of your normal AUTOEXEC.BAT (on C:/) that makes C: the default drive:

```
C:
```

That's all it takes. Then, write a new one-line AUTOEXEC.BAT for drive A: that transfers command to drive C: and your normal AUTOEXEC:

```
C:AUTOEXEC
```

If you already are using one of the more powerful memory managers and have your system configured for its use, it generally is better, I find, to work with an alternative startup batch file. I call mine ALTSTART.BAT. It is similar to my working AUTOEXEC.BAT, except it has any references to proprietary programs designed to work in conjunction with my normal memory manager to load TSRs above 640K. For instance, in the following case, the portion shown in italics is deleted in the ALTSTART.BAT, leaving the basic commands themselves:

```
dos\qemm\loadhi /r:1 dos\qemm\lastdrive=Z
dos\qemm\loadhi /r:1 dos\qemm\files +20
dos\qemm\loadhi /h DOS\MODE COM1:9600,N,8,1,P
dos\qemm\loadhi /h DOS\MODE COM2:9600,N,8,1,P
dos\qemm\loadhi /r:2 DOS\PCKWIK\SUPERPCK / S:1000
```

There are two command lines that must be removed altogether, requiring, in this case, that the FILES and LASTDRIVE functions be set from DOS in the CONFIG.SYS, which would normally be the case anyway. In the other cases, only the parts of lines dealing with LOADHI have had to be removed.

Using this ALTSTART.BAT as a template that can be copied to bootable floppies as a starting point eliminates the need for disturbing my normal operating configuration until—and unless—I have determined experimentally that what I am trying is better—and until I'm reasonably sure I've got the bugs worked out of the configuration. To use the alternate startup template, I simply copy it to a new bootable floppy, changing the name to AUTOEXEC.BAT in the process:

```
C>COPY ALTSTART.BAT A:AUTOEXEC.BAT
```

A basic template for a starting CONFIG.SYS to use on new bootable floppies can be created and used in much the same way.

Many users find one of the most objectionable features to booting from a floppy is the fact that DOS automatically sets COMSPEC= to the boot drive. This means that every time DOS needs COMMAND.COM it has to go back to drive A:, which is not only slow but prevents you from using A: for almost anything else that session. You can get around this problem easily. If you are not changing DOS versions (or trying an alternate operating system), all you have to do is include a COMSPEC statement in your floppy AUTOEXEC:

```
SET COMSPEC=C:\COMMAND.COM
```

This simply points to a copy of COMMAND.COM on the hard disk root directory—the one you probably would normally use anyway. However, COMMAND.COM (or an alternate command interpreter, such as 4DOS) can reside almost anyplace you'd like to have it. This opens up several possibilities of particular interest when upgrading to a new DOS version or trying a different operating system.

For example, you might use a COMSPEC statement something like this:

```
SET COMSPEC=C:\DR_DOS\COMMAND.COM
```

This points to a subdirectory on C: for the command interpreter for Digital's DR DOS. You can either copy the utilities associated with that operating system to that subdirectory and PATH to the subdirectory or you can keep them on your bootable floppy (and PATH to the floppy). This latter scheme will require you to have the disk containing these files in drive A: whenever they are needed.

Alternately, you can copy the command interpreter for the operating system to a RAM disk and set your COMSPEC to point there. I prefer this approach. I usually copy some of the most often used version-specific utilities to the RAM disk, as well. Things run even faster this way than if these files resided only on the hard disk.

Finally, be sure that you label your bootable floppies carefully so you can keep track of them for future use—and possibly for further experimentation. I use an inexpensive program called NAME THAT DISK, which reads the contents of each disk and prints labels either with specific filename or wildcards, along with pertinent data. Whether you choose to follow these suggestions or go it your own way, the most important thing is to have some kind of system and to stick with it.

As a final note, there are a few proprietary installation programs that, in an effort to make themselves idiot-proof, do not allow installation except to your hard disk. They cavalierly make changes to your CONFIG.SYS and AUTOEXEC.BAT pretty much to suit themselves. I absolutely hate such programs and generally try to give them a wide berth. However, not all of them give you adequate warning in advance—something I hate even worse. To prevent the loss of my working CONFIG.SYS and AUTOEXEC.BAT, I always make sure I have an updated copy of both files saved under different names, so I can restore them quickly, if necessary.

D

APPENDIX

The basics of hexadecimal

This book is about addresses—computer addresses. Everything a computer does involves some block of addresses. Devices (printers, keyboards, and monitors) are installed at addresses. Software installs itself at addresses. For the purposes of this book, there are about a million of them. To be precise, there are 1,048,576 unique and identifiable absolute addresses—or, in simpler terms, FFFFh (the "h" is for "hexadecimal"). Using hexadecimal is much simpler than using powers and multiples of two.

I don't have to go deeply into hexadecimal—and don't intend to here. A few of the basics however, will make life easier.

People count in tens (the decimal system) because its easy—for them at least. Computers count in twos (the binary system) because that's all they've got. Each transistor (or equivalent speck buried in a chip) is either switched on (0) or off (1). They, however, count a lot faster and better than people do in tens. Like it or not, at some point we have to come to terms with it, but not necessarily by adding up long strings of twos.

As you can quickly see, the number 16 (2^4) provides a convenient compromise. A bunch of impossible-to-remember binary numbers is equated to a neat uniform progression in hexadecimal. The first ten blocks (0 through 9) cover the entire 640K of conventional user memory available to ordinary applications.

This is good so far; however, six more single-character somethings are needed to bring the count to 16. The letters: A, B, C, D, E, and F are used to fill in the gap. Now the count runs 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Referring again to Fig. D-1, you'll see the progression in either notation is just a series of blocks of 64K. These blocks really aren't nice, neat increments of 64,000 bytes, but actually 65,536. In hexadecimal, it really is a nice, neat, even number: 10000h. That's really all there is to understanding hexadecimal. At least, it is enough to start putting hexadecimal numbers to work for you.

In various literature find two address-numbering conventions used. Both conventions appear to be in hex notation, with a mix of letters and numbers; however, some will have only four characters, while others will have five.

Even in hex, it takes five places to show the whole of addresses to 1024 kilobytes (1 Mb). To be precise, it takes 20 bits to point to any address in that range. The 8088, however, has only 16-bit address registers to work with. As a result, addresses must be broken up into components that can be managed by 16-bit registers. The result is a two part address, consisting of a four-character "segment" (*cccc*:) and a second four character offset group. The resulting address looks something like:

xxxx:*yyyy*

To arrive at the full 32-bit address used in real mode from these two 16-bit numbers, you add these two values, offsetting the second part in this manner:

```
 1111
+ 2222
13332
```

For the purposes of this book, however, the segment address (like any number rounded off for the sake of convenience) the segment address contains all of the information needed here, unless otherwise noted.

In this book, I will generally refer to blocks of memory—64K for instance—in their decimal approximations, as also is commonly done. The most significant break points in DOS still come at multiples of 65536 bytes—multiples of 10000 in hexadecimal.

What is 3 times 65536? The answer is 30000 in hexadecimal, or 3000 if you drop a zero to reduce the number to a four digit segment. Instead of complicating life, hex really makes counting easier when dealing with computers.

E
APPENDIX

Addresses for the software developers

The list below contains the addresses and phone numbers of many of the software developers and distributors mentioned in this book:

AI Architects, Inc.
One Kendall Square
Cambridge, MA 02139
(617) 577-8052

Rational Systems, Inc.
P.O. Box 480
Natick, MA 01760
(508) 653-6006

Phar Lap Software, Inc.
60 Aberdeen Avenue
Cambridge, MA 02138
(617) 661-1510

Intelligent Graphics Corp.
4800 Great America Parkway
Santa Clara, CA 95054
(408) 986-8373

Advanced Logic Research, Inc.
9401 Jeronimo
Irvine, CA 92718
(800) 444-4ALR

Rose Electronics
P.O. Box 742571
Houston, TX 77274

ALL Computers, Inc.
34711 Chardon Road
Willoughby Hills, OH 44094
(216) 944-0110

Buffalo Products
P.O. Box 117097
Burlingame, CA 94011
(800) 345—BFLO

Digital Research, Inc.
Box DRI
Monterey, CA 93942
(408) 649-3896

The Software Link, Inc.
3755 Parkway Lane
Norcross, GA 30092
(404) 448-5465

Hauppauge Computer Works, Inc.
91 Cabot Court
Hauppauge, NY 11788
(516) 434-1600

Distributed Processing Technology
132 Candade Drive, P.O. Box 1864
Maitland, FL 32751
(305) 830-5522

Paul Mace Software
400 Williamson Way
Ashland, OR 97520

Newer Technology
1117 South Rock Road, Suite 4
Wichita, KS 67207
(316) 685-4904

I discussed many software products and packages throughout this book. The following list contains several of the pieces of software that I covered more thoroughly:

Turbo EMS

Merrill and Bryan Enterprises, Inc.
9770 Carroll Center Road, Suite C
San Diego, CA 92126

System Sleuth

Dariana Technology Group, Inc.
7439 La Palma Avenue, Suite 278
Buena Park, CA 90620

HeadRoom

Helix Software
83-65 Daniels Street
Briarwood, NY 11435

Printer Genius

Nor Software, Inc.
P.O. Box 1747
Murray Hill Station
New York, NJ 10156

PrintQ

Software Directions, Inc.
1572 Sussex Turnpike
Randolph, NJ 07869

hDC FirstApps

hDC Computer Corp.
6742 185th Avenue N.E.
Redmond, WA 98052

HOPTIMUM

Paul Mace Software
400 Williamson Way
Ashland, OR 97520

Software Carousel

Softlogic Solutions
530 Chestnut Street
Manchester, NY 03101

Glossary

alternate register set A single register set can only point to a single set of 16K blocks (logical pages) in expanded memory. To access more than 64K of data (under 3.2 EMS) or provide one set of 16K blocks to backfill conventional memory and/or access data in expanded memory (under 4.0 EMS), additional sets of pointers—alternate register sets—must be provided.

ASCII American Standard Code for Information Interchange. A method of translating computer machine language to a user-identifiable character set.

beta A final phase of testing where a product is put in the hands of selected users for a period of actual in-use testing prior to general distribution.

BIOS Basic In/Out Services. A set of instructions managing computer operations at the lowest level.

boot sector A special disk sector reserved for information that the computer needs to load the operating system.

byte The basic unit of measure for computer memory. A character (such as a letter, number, or punctuation mark) uses one byte of memory. A byte is composed of eight binary bits.

conventional memory The term applied to user memory, generally below 640K, but under certain circumstances can be increased to 704K or more depending on the type of display and video addresses used.

configuration The grouping of hardware and software that make up a computer system. The grouping includes the main console (display), any printers, the operating system, and any other applications and hardware.

data compression A means of packing data to lessen the space it occupies for storage.

device driver A piece of software that contains the specifications for running a particular device. When invoked, it activates and controls communications with the device.

DOS Derived from Disk Operating System. It is sometimes used generically, but in this book is used specifically to refer to the Microsoft operating systems marketed as IBM DOS (sometimes called PC DOS) and MS-DOS (often renamed by vendors as COMPAQ DOS, TANDY DOS, etc.).

EEMS Enhanced Expanded Memory Specification. An enhanced version of the earlier LIM 3.2 EMS. A superset of EMS, many features were incorporated in LIM 4.0 EMS.

executable file A file that contains machine-recognizable language that can be executed directly without an interpreter.

expanded memory Nonlinear memory beyond 1 Mb that is accessible on revolving basis in blocks made available by an Expanded Memory Manager (EMM) at addresses within DOS's 1 Mb limits.

extended memory Linear memory at addresses above 1 Mb, it is accessible and directly usable only with 80286 and higher processors and can be used to emulate expanded memory.

gigabyte One billion (10^9) bytes.

hexadecimal The base-16 numbering system derived from the binary nature of computer logic (2^4), which cannot deal directly with decimal values.

high DOS memory The term used to refer to RAM mapped to unused address spaces above 640K but below 1024K. It also is referred to as Upper Memory Blocks (UMBs).

high memory area The "extra" 64K available to DOS between 1024K and 1088K, applicable with 286 and higher CPUs only.

kilobyte One thousand (10^3) bytes.

LIM Acronym derived from Lotus/Intel/Microsoft, used to describe Expanded Memory Specification (EMS), which was developed through a joint effort that ultimately included other firms as well.

linear memory Directly accessible memory made up of conventional memory (including upper memory from 640K to 1 Mb) and extended memory above 1 Mb for 80286- or 80386-based machines.

logical page 16K block of memory. Under 3.2 EMS, it applied only the blocks (four per 64K page frame) above 640K. Under 4.0 EMS, it can apply to any 16K block (with a base address that is a multiple of 16K) below 1024K.

low RAM Conventional RAM.

megabyte One million (10^6) bytes.

multitasking Running multiple applications simultaneously. With DOS, this is facilitated by using a multitasking or windowing environment, such as Microsoft Windows or DESQview. Protected mode operating systems, such as Xenix and OS/2, have this capability integral to them.

nanosecond One-billionth of a second.

overhead The memory used by whatever software is loaded. The operating system requires a substantial amount of memory and still more is added by any TSRs, etc.

page A 16K block of information that can be electronically repositioned from expanded memory to conventional memory through the bank switching process.

page frame An area through which pages are switched in the bank switching process.

page register A memory location that acts as a point and locates a particular area in memory.

paged memory Memory that is divided into 16K blocks called pages. (*see* expanded memory)

perfect superset A term used to describe some set of functions or commands that go beyond the limitations of the underlying system to provide additional services, while maintaining absolute compatibility with same. EEMS, for instance, was a perfect superset for the original LIM EMS specification.

port The physical device that serves as a channel for peripheral devices to communicate with the Central Processing Unit (CPU).

print spooler An area in volatile memory (RAM) set aside to take data that is being sent to the printer, thereby freeing the processor for other tasks. While not considered multitasking in the usual sense, it is to the extent that the user can go on with some other task even while the job is still printing.

protected mode A special mode of operation that allows addressing of up to 16 Mb of extended memory with 80286 systems. It currently is available in alternative operating systems, such as Xenix and OS/2. It also can be used by the 80386.

RAM Volatile memory. An area of memory in which code and data can be stored during processing.

RAM disk An area of volatile memory (RAM) set aside for the quick access of data. When data is placed in a RAM disk, a program can access it much more quickly than it could if it had to read that same data from a floppy or hard disk.

register The pointer required to locate a particular 16K block of data stored in expanded memory.

register set The collection off all of the registers used by a particular program or application at any given time.

remapping The ability to assign unused addresses (typically above 640K) to blocks of physical RAM, so they appear to be at those addresses. It generally is limited to 80386 systems where it is supported by the microprocessor chip itself.

ROM BIOS Low-level Basic In/Out Services loaded into memory during boot processes from Read-Only Memory chips.

SIMM Single Inline Memory Module A mini circuit board generally containing one complete bank of typically nine memory chips. This type of memory unit takes up significantly less space than individual DRAM chips in sockets but requires special socketing unique to this type of installation.

Split Memory Addressing A term used by AST to define the ability of its boards to allocate some of their memory to fill out conventional memory (up to 640K), allocating the rest to expanded or extended memory.

superset A term used to describe some set of functions or commands that go beyond the limitations of the underlying system to provide additional services. (*see* perfect superset)

terabyte One trillion (10^{12}) bytes.

time slicing A commonly used technique for switching (dividing the attention of) a single microprocessor chip between two or more applications. If switched fast enough, it gives the illusion of simultaneous processing, or multitasking.

thrashing Where the time slice alloted to an application using virtual memory in a multitasking environment is insufficient to complete read or write access. As long as that access remains incomplete, the heads thrash back and forth each time that application's time slice comes up until the task is finished.

upper memory Sometimes called reserved memory, this is any memory in the 640K to 1 Mb address range. It originally was used for system functions, such as display memory, ROM BIOS, and various auxiliary functions. Now it also includes page frames used by expanded memory and, with remapping (80386 only, except with additional hardware support) is used for relocation of various functions from conventional 640K.

Vdisk An IBM software product that creates a simulated disk drive in RAM.

volatile A term used to describe memory that retains its contents only as long as it is receiving power to refresh itself. Common RAM falls into this category.

XMS A memory usage and management specification written by Microsoft (but also incorporating the work of others) that defines a protocol controlling access to high (1024K to 1088K), upper (640K to 1024K), and extended memory on all computers using 80286 or higher chips.

Index

A

oveBoard, 55, 59, 238
celerator cards, 207, 208-212
  access time, 209
  ALL 386SX, 210-211
  bootup time, 209
  InBoard 386PC, 208-210
  SnapIn 386, 211-212
dressing memory, 2-5, 38-39
  80386 microprocessor
    capabilities, 25
  segment and offset, 2-4
Architects, 97, 98
L 386SX accelerator card, 210-211
l Computer, 83, 115, 120-123, 170, 172, 212
LCharge 386, 71, 75, 172, 176, 243
LMENU, 80, 83, 245
ocation,memory (*see* memory management)
R Inc., 231
NSI display driver, 12
T Research, 39-41, 130, 232
  DESQview, 153-154
  EMS memory, 63-64
toCAD, 89
TOEXEC.BAT
  ootable floppies, testing new configurations, 265-267
  DR DOS, 136, 137
  testing new configurations on bootable floppies, 265-267
  virtual machines, 187-188

B

backfilling, conventional memory mapping, 58-59
BASIC, 118
BASICA, 118
BAT files, 14
BIOS (*see* ROM BIOS)
BlueMAX, 78, 118-119, 243
bootable floppies, new-configuration testing, 265-267
bootup, triple fault, rebooting, 22
Borland, 97, 98
Buffalo Products, 151
buffers, print, 150-151
buses
  direct memory access (DMA), 229
  EISA, 227-228, 230-231
  EMS memory, speed, 65-66
  micro channel architecture (MCA), 226-229
  PC/AT, speed considerations, 228

C

caching disks, 219-220
Carousel (*see* Software Carousel)
ChargeCard, 71, 83, 115, 120, 122, 123, 212-214, 223, 238
CheckIt, 241
Chips and Technology, 123, 135, 138
clock speed, 4, 31-33
  buses, PC/AT, 228
  dynamic vs. static RAM, 32
  EMS memory, 65-66
  polling, 33
  time slicing, 32-33
  upgrading considerations, 223
  virtual machines, 181
  wait states, 31
COM files, 14
COMMAND.COM, 13-14
commands, internal vs. external, 14
compatibility method (MR:C), 120
compatibility, DOS versions, 4-5
CONFIG.SYS
  bootable floppies, testing new configurations, 265-267
  DR DOS, 136, 137
  EMM386.EXE, 71-74
  testing new configurations on bootable floppies, 265-267
  virtual machines, 187-188
configuration
  bootable floppies, testing new configurations, 265-267

configuration (*cont.*)
  on-the-fly, DR DOS, 137
  virtual machines, 185-187
context switching, Software Carousel, 145-146
coprocessors, 30-31
CP/M, 18
crashes, software and memory management, 246-249

D

DC Windows Express, 262
DEBUG, 249-251
Deluxe Paint, 262
demand-paged virtualization, DESQview, 162
descriptors, protected mode, 88
DeskLink, 148
DESQview, 19-20, 42, 45, 54, 56, 82, 95, 98, 104, 105, 106, 121, 129, 136, 137, 148, 153-167, 192, 240, 243, 247
  AST Research, 153-154
  backup program (Sitback), 159-160
  demand-paged virtualization, 162
  DOS extenders, 163
  enhanced expanded memory specification (EEMS), 153-154
  high memory area (HMA), 108, 160
  LIM 4.0 EMS, 154-157
  macros, scripts, 165-166
  memory allocation and use, 158
  multitasking, 153
  path statements, 158-159
  PIF files, 164-165
  QEMM, 160-162
  virtual control program interface (VCPI), 163
  Windows, 163-164
  X Window System, 166-167
device drivers (*see* TSRs)
DEVICEHIGH, 71
Digital Research, xviii
direct memory access (DMA), 229
DOS, 10-12
  ANSI display driver, 12
  BIOS module, IBMBIO.COM, 13
  COMMAND.COM, 13-14
  evolution and development, 12-13
  expanded memory use, 14
  extenders, 21, 25-26, 90-92, 95-98, 254-255
  IBMDOS.COM, 13
  internal vs. external commands, 14
  ROM BIOS, 11-13
  shell, COMMAND.COM, 13-14
DOS protected mode interface (DPMI), 21, 274
  80386 support, 99-101
  extended memory, 99-101
  Lotus, 95
  Windows, 170
DR DOS, xviii, 127-128, 134-140, 253
  AUTOEXEC.BAT, 136, 137
  CONFIG.SYS, 136, 137
  configuration-on-the-fly, 137
  EMM386.SYS, 135
  EMMXMA.SYS, 136
  HIDOS.SYS, 135-136, 138
  high memory area (HMA), 108, 134, 135
  HINSTALL, 136-137
  memory management, MEM, 114, 240, 245
  multiuser options DR Multiuser DOS, 195-200
DR Multiuser DOS, 195-200
dynamic RAM, 32, 226

E

80286 CPU chip, xviii-xix, 15, 19, 20-23
  addressing capabilities, 4
  clock speed, 4
  DOS extenders, 21
  extended memory development, 20, 41-42, 85, 90
  LOADALL function, 22
  memory management, 83
  memory-mapping, 67, 83
  protected mode, 42, 87, 88
  real vs. protected mode use, 20
  triple fault, 22
80386 CPU chip, xvii-xviii, 19, 23-29
  addressing capabilities, 4
  clock speed, 4
  compatibility with other microprocessors, 24
  DOS extenders, 25-26
  DPMI support, 99-101
  DX, 80386 DX developme and use, 27, 30
  EISA bus, 27
  expanded memory, 45
  extended memory, 25, 85,
  memory-mapping and addressing, 25, 67
  Micro Channel Architectur
  protected mode operations, 88
  real mode operation, 25
  SL, 80386 SL, 29
  SX, 80386 SX, 27-28
  VCPI support, 98-101
  virtual machine, 26
  virtual mode, 26
8086 chip, 17-19
8088 chips, xviii, 15, 16-20
  expanded memory development, 19
  memory-mapping, 67
  protected-mode use, 19
EMM386.COM, 101
EMM386.EXE, 64-65, 71-7 101, 114, 115, 117, 128, 130-134, 138, 171-173, 2 245
EMM386.SYS, 134, 135, 13
EMMXMA.SYS, 136
EMS memory, xviii, 55-66
  add-in memory boards, 60
  AST Research developmen 55, 63-64
  backfill, conventional mem mapping, 58-59
  bus speed, 65-66
  EMS emulators, EMM386.EXE, 64-65
  enhanced expanded memo specification (EEMS), 55
  high memory area, 58-59
  HIMEM.SYS, 65
  LIM 4.0 EMS, 56-64
  mapping registers, 60

  memory-management, 64
  multitasking, 56
  register sets, 61-62
  registers, 60-65
  wait states, 66
ulation, expanded memory, 53-54
ulators, expanded memory, 53-54
hanced expanded memory specification (EEMS), 39, 42-43, 55-57, 153-154, 231-232
hanced mode, Windows, 171
go Computers, 97
xcel, 262
XE files, 14
panded memory, 19, 35, 47-54, 67
  80286 and extended memory, 41-42
  80386 development, 45
  allocation of memory, 48-50
  AST Research development, 39-41
  disabling system memory, 44-45
  DOS device drivers, 14
  emulation vs. emulators, 53-54
  emulation, MS-DOS 5.0 and DR DOS, xviii
  emulators, 53-54
  enhanced expanded memory specification (EEMS), 39, 42-43, 55-57
  expanded memory, 48-50
  extended memory vs., 86
  high memory area use, 49
  LIM EMS 3.2, 39, 42
  LIM EMS 4.0, 37, 43-44, 47
  ogical pages, 50-51
  Lotus development, 48
  memory-management, 40-41, 52-53, 115, 239
  Microsoft development, 48
  multitasking, 39, 40
  page frames, 36, 49-52
  pages, accessing pages, 36
  RAM allocation, 43
  Turbo EMS, 53-54
  virtual machines, 180, 184
panded memory (EMX), xviii
expansion boards, 233-235
extended industry standard architecture (EISA)
  80386 DX, 27
  buses and upgrading, 227-228, 230-231
  i486 chip, 29
  upgrading, motherboard replacement, 215
extended memory, 20, 25, 85-101
  80286 development, 41-42, 85, 90
  80386, 85, 90
  accessing schemes, 89-90, 104
  address pin limitations, 86-87
  continuous memory, 86
  DOS extenders, 90-92, 95-98
  DOS protected mode interface (DPMI), 99-101
  expanded memory vs., 86
  extended memory specification (XMS), 90
  high memory area (HMA), 90
  i486, 86, 90
  Lotus development, 92-95
  memory-management, 45, 114
  memory-mapping, 67
  OS/2 compatibility, 95
  protected mode, 42, 87-89
  upper memory blocks (UMBs), 90
  VDISK, 42, 89
  virtual control program interface (VCPI), 98-101
  virtual machines, 180, 184
extended memory manager (XMM), high memory area (HMA), 106
extended memory specification (XMS), 90, 106
extenders, DOS, 21, 25-26, 90-92, 95-98, 254-255
  DESQview use, 163
  DOS protected mode interface (DPMI), 21, 274
  virtual control program interface (VCPI), 21, 274

F

4.0 LIM EMS, 232
4004 chip, Intel, 16-17
FAX boards, 148-149
floppy disks, bootable, new-configuration testing, 265-267
fragmentation, 76-78
functions, LIM 4.0 EMM, advanced programming functions, 257-259

G

Global Descriptor Table (GDT), 88
GWBASIC, 118

H

hard disks
  disk caching, 219-220
  interleave factor, 217-219
  optimization software, 218-219
  RAM disks, 219-220
hDC FirstApps, Windows, 176-178
Headroom, 114, 142-145
hexadecimal number system, 269-270
HIDOS.SYS, 109, 135-136, 138
high memory area (HMA), xix, 44, 49, 58-59, 103-111, 115
  accessing schemes, 104-106
  addressing and segments, 104-106
  DESQview, 108, 160
  DR DOS, 108, 134, 135
  extended memory, 90
  extended memory manager (XMM), 106
  extended memory specification (XMS), 106
  HIDOS.SYS, 109
  HIMEM XMS, 109
  HIMEM.SYS, 106, 108
  loading files/programs, 107
  memory allocation and use, MEM, 109-111
  memory management, 114, 116
  memory-mapping, 67
  MS-DOS, 128-129
  QEXT, 106
  VDISK, 104, 106, 107
HIMEM.SYS, 71-74, 101, 106, 108, 115-117, 128-129, 138, 170, 172-173, 180, 240, 245
  EMS memory, 65

- HIMEM.SYS (*cont.*)
  - memory management, 114
- HIMEM.XMS driver, 109
- HINSTALL, 136-137
- HOPTIMUM, 218-219

I

- i486 chip, xvii-xviii, 15, 29-30, 221-222
  - clock speed, 4
  - EISA bus, 29
  - extended memory, 86, 90
  - Micro Channel Architecture, 29
  - SX, 30
- i860 chip, 31, 221
- IBMBIO.COM, 13
- IBMDOS.COM, 13
- IDLE feature, DR Multiuser DOS security, 199-200
- InBoard 386PC accelerator card, 208-210
- InfoSpotter, 80, 240, 245, 249-250
- Intelligent Graphics Corporation, 192
- Interleaf Publisher, 97
- interleave factor, 217-219

J

- Javelin, 262
- JLASER/SA, 262

L

- laptop computers, 224-225
- LeAPSet, 135, 138
- LeAPSetsx, 135, 138
- LIM 3.2 EMS, 39, 42, 55, 59
- LIM 4.0 EMS, 19, 37, 43-44, 47, 56-58, 59-64, 83, 115, 117, 128
  - advanced programming functions, 257-259, 261
  - bus speed, 65-66,
  - DESQview, 154-157
  - memory-mapping, 67, 261-263
  - MS-DOS, 130
- LOADALL function, 22
- LOADHIGH, 71
- Local Descriptor Table (LDT), 88
- logical pages, expanded memory, 50-51
- Lotus, 97
  - DPMI, 95
  - expanded memory development, 48
  - extended memory development, 92-95
  - VCPI, 95

M

- macros, DESQview, 165-166
- Manifest, 79, 80, 124, 240, 243-245, 251
- Map, 76
- mapping (*see* memory-mapping)
- mapping method (MR:M), 120
- mapping registers, EMS memory, 60
- Maximize, 75, 78, 107, 139
- MEM command, 73, 249, 251
  - high memory area (HMA), 109-111
  - memory management, 240, 245
- memory allocation
  - above 640K, user vs. system, 7-9
  - addressing, 2-5
  - addressing, segment and offset, 2-4
  - assignment of memory, user vs. system, 5-9
  - backfilling, conventional memory mapping, 58-59
  - contiguous memory blocks, above 640K, 9
  - DOS extenders, 254-255
  - dynamic RAM vs. static RAM, 32
  - enhanced (*see* enhanced memory)
  - expanded (*see* expanded memory)
  - high area (*see* high memory area)
  - LIM 4.0 EMS (*see* LIM 4.O EMS)
  - managing (*see* memory management)
  - reserved, xix
  - video usage, 7-9
- memory boards, 66
- Memory Commander, 120-122
- memory management, 37, 69-7
  79-81, 113-125, 237-251
  - 386MAX, 113, 116, 240, 24
  - 80286, 83
  - AboveBoard, 238
  - ALLMenu, 245
  - BlueMAX, 118-119
  - ChargeCard, 122, 123, 212-214, 238
  - CheckIt, 241
  - compatibility method (MR:C
    120
  - DEBUG, 249-251
  - DESQview, 121
  - device drivers, 114
  - DR DOS, 114, 240, 245
  - EMM386.EXE, 71-74, 114, 117, 240
  - EMS memory, 64
  - expanded memory, 40-41, 52-53, 115, 239
  - extended memory, 45, 114
  - fragmentation, 76-78
  - Headroom, 114
  - high memory area (HMA), 114, 115, 116
  - HIMEM.SYS, 71-74, 114, 116, 117, 240
  - InfoSpotter, 240, 245, 249-2
  - large files, 75-76
  - LIM 4.0 EMS, 115, 117
  - Manifest, 124, 240, 243-245
    251
  - mapping method (MR:M), 1
  - MEM command, 73, 240, 2
    249, 251
  - Memory Commander, 122
  - MOVE'EM, 83, 123
  - MS-DOS, 240, 245
  - multiuser options, 205
  - NEAT, 123
  - Optimize, 114
  - PopDrop, 114
  - ports, parallel vs. serial, 246
  - protected method (MR:P) mapping, 120
  - PS/2, 119
  - QEMM, 113, 116
  - QEMM50/60, 119
  - QRAM, 123
  - reserved memory, 114

ROM BIOS, 78-79, 118-119
cram!, 119, 120-121
ID, 249
leuth, 241
oftware crashes, 246-249
queeze!, 121
SRs, 75, 80, 81, 114
ideo RAM, 68, 81-83,
123-135, 239-240
irtual machines, 180
M386, 121
Vindows, 121, 170-173
mory Viewer, Windows,
177-178
mory-mapping, 67-83
0286, 67, 83
0386 microprocessor
capabilities, 25, 67
088, 67
ackfill, 58-59
ompatibility method (MR:C),
120
MM386.EXE, 71-74
xpanded memory, 67
xtended memory, 67
ragmentation, 76-78
ree or unassigned memory, 69
igh memory area (HMA),
MEM, 67, 109-111
rge files, 75-76
IM 4.0 EMM, 67, 261-263
apping method (MR:M), 120
IEM command, 73
emory management, 69-71
IOVE'EM, 83
rotected method (MR:P)
mapping, 120
OM BIOS, 78-79
ideo RAM, 68, 81-83
irtual machines, 184
rrill and Bryan, 240
ro channel architecture
MCA)
0386 DX, 27
uses and upgrading, 226-229
86 chip, 29
roprocessors, 15-33
de, 76
herboard replacement,
pgrading, 214-217
VE'EM, 83, 123

MS-DOS, xviii, 127-134, 253
  EMM386.EXE, 130-134
  HIMEM.SYS, 128-129
  LIM 4.0 EMS, 130
  memory management, MEM, 240, 245
  Microsoft development, 128-130
  Windows, 129
multitasking, 100
  DESQview, 153-167
  EMS memory, 56
  expanded memory, 39, 40
  Quarterdeck, 40
  virtual machines, 181-183
multiuser DOS (MDOS), xviii, 191-205
  DR DOS, 195-200
  virtual machines, VM386, 192-195
  VM386, 192-195
multiuser options
  DR Multiuser DOS, 195-200
  memory management, 205
  multiuser DOS (MDOS), 191-205
  PC-MOS, 200-204
  SCO VP/ix, 204
  virtual machines, 189-190
  XENIX, 204

N

NEAT, 123, 135, 138
NEATsx, 135, 138
New Technology, 66
number systems, hexadecimal vs. binary and decimal, 269-270

O

offset, address, 2-4
operating systems, 5, 10-12, 100
Optimize, 75, 78, 107, 114, 121, 139
Oracle, 95
OS/2, 5, 11, 19, 25, 86, 89, 90, 95, 100, 118
  Lotus compatibility, 95

P

Packrmd, 76-77
page frames, expanded memory, 36, 49-52
Pagemaker, 262
pages, expanded memory, 36
Paradox, 97, 98, 262
parallel ports, 246
PC-MOS multiuser option, 11, 200-204
  bootup procedures, 203-204
  compatibility, 202-203
  device driver loading, 202
  installation, 203-204
peripheral sharing, 151
Phar Lap Software, 36, 97, 98
polling, 33
PopDrop, 114
ports, parallel vs. serial, 246
PRINT, 150
print spoolers, 149-150
protected method (MR:P) mapping, 120
protected mode, 19, 20, 25
  80286 operations, 87, 88
  80386 operations, 88
  descriptors, 88
  extended memory, 42, 87-89, 97
  Global Descriptor Table (GDT), 88
  Local Descriptor Table (LDT), 88
  selectors, 88
PS/2, 119

Q

QEMM, 68, 70, 75, 100, 104, 107, 113-116, 120, 121, 124, 135, 136, 137, 139, 160-162, 173-174, 180, 240, 242, 243, 253
QEMM.SYS, 172
QEMM386, 78, 82, 119
QEMM50/60, 119
QEXT, 106
QRAM, 123, 139, 240, 242, 243
Quadram, 98
Qualitas, 45, 71, 75, 78, 83, 98, 100, 118-119, 121, 123, 170, 172
Quarterdeck, 36, 40, 42, 45, 56, 71, 78, 90, 98, 100, 104, 105, 106, 119, 121, 123, 135, 170, 172, 240, 253

R

RAM allocation, expanded memory, 43
RAM disks, 53, 219-220
Rampage (*See also* AST Research), 39-41, 55, 64, 130
Rational Systems, 97, 98
real mode, 20, 25
real mode, Windows, 171-172
reduced instruction set computing (RISC) chips, 31
register sets, EMS memory, 61-62
registers, EMS memory, 60-65
reserved memory area, xix, 114
ring 0, DPMI, 99
ROM BASIC, 10, 118
ROM BIOS, 11-13, 78-79, 118-119
Rose Electronics, 151

S

SCO VP/ix, 204
Scram!, 119, 120-121
scripts, DESQview, 165-166
segment, address, 2-4
selectors, protected mode, 88
serial ports, 246
SHELL command, 13
  task swapping, 146-148
shell, DOS, 13
SID, 249
SIMM chips, 225-226
Sleuth, 80, 241
Smartware II, 262
SnapIn 386 accelerator card, 211-212
software developers' addresses, 271-273
Software Carousel, 19, 39, 54, 56, 145-146
SQL Windows, 262
Squeeze!, 121
Squish, 76
Stacker, 262
standard mode, Windows, 171-172
static RAM, 32, 226
SuperCalc, 262
Superpck, 76-77
SuperProject+, 262
System Sleuth, 80, 241

T

386EMM.SYS, 14
386MAX, 45, 68, 70, 75, 79, 100, 107, 113, 115, 116, 136, 137, 139, 172, 174-176, 240, 243, 245
task swapping, SHELL, 146-148
task switching, 148
  HeadRoom, 142-145
time slicing, 32-33
Traveling Software, 148
triple fault, 22
TSRs, memory management, 75, 80, 104, 113, 114
Turbo EMS, 53-54

U

UNIX, 12, 100
upgrading, 207-235
  4.0 LIM EMS, 232
  8-bit vs. 16-bit expansion boards, 233-235
  80286 vs. 80386 upgrades, 222-224
  80286-to-80386 upgrades, 210
  accelerator cards, 207-212
  ALL 386SX accelerator card, 210-211
  buses, direct memory access (DMA), 229
  buses, EISA, 227-228, 230-231
  buses, micro channel architecture (MCA), 226-229
  buses, PC/AT, speed considerations, 228
  ChargeCard, 212-214
  clock speed considerations, 223
  CPU replacement, 207
  disk caching, 219-220
  dynamic vs. static RAM, 226
  enhanced expanded memory specification (EEMS), 231-232
  hard-disk optimization, HOPTIMUM, 218-219
  i486 chips, 221-222
  i860 chips, 221
  interleave factors, 217-219
  laptop computers, 224-225
  microprocessor chip replacement, 221-235
  motherboard replacement, 214-217
  RAM disks, 219-220
  recycled hardware, 233
  SIMMs, 225-226
  SnapIn 386, 211-212
  SX vs. DX upgrades, 223-
upper memory blocks (UMB
  extended memory, 90

V

V Communications, 122
VDISK utility, 42, 89, 104, 1
  107
Ventura Publisher, 262
video RAM, 7-9
  memory management, 68, 81-83, 123-125, 239-240
  memory-mapping, 68, 81-
VIDRAM.COM, 82
virtual control program interf
  (VCPI), xviii, 21
  80386 support, 98
  DESQview, 163
  extended memory, 98-101
  Lotus, 95
  Windows, 95, 170
virtual machines, 26, 99, 10
  179-190
  addressing memory, 183-1
  AUTOEXEC.BAT customization, 187-188
  booting procedures, 179-1
  clocks, clock speed, 181
  compatibility, 8086-8088 machines and up, 180
  CONFIG.SYS customizati
  187-188
  crashes, 188-189
  environmental configuratio
  185-187
  expanded memory, 180, 18
  extended memory, 180, 18
  HIMEM.SYS, 180
  mapping memory, 184
  memory management, 180
  multitasking, 181-183
  multiuser options, 189-190
  QEMM, 180
  VM386, 180-190
virtual memory, EMS emulat
  53-54

ual mode, 26, 179
386, 121, 137, 180-190
ultiuser DOS (MDOS),
192-195
ix, 204

W

t states, 31, 66
ndows, xviii, 19, 35, 56, 70,
121, 129, 169-178, 247
86MAX, 172, 174-176
LLCharge386, 172, 176
DESQview, 163-164
DOS protected mode interface (DPMI), 170
EMM386.SYS, 172-173
enhanced mode, 171
hDC FirstApps, 176-178
HIMEM.SYS, 172-173
memory management, 170-173
Memory Viewer, 177-178
QEMM, 172-174
real mode, 171-172
standard mode, 171-172
VCPI incompatibility, 21, 95, 170
versions, differences, 171-172
X Window System, 166-167
Word for Windows, 262

X

X Window System, 166-167
XENIX, 11, 204
XMA2EMS.SYS, 14
XMAEM.SYS, 14